# RESCUE: Evaluation of a Fragmented Secret Share System in Distributed-Cloud Architecture

**Elochukwu Anthony Ukwandu**

**Matriculation Number:** [redacted]

**A thesis submitted in partial fulfilment of the requirements of Edinburgh Napier University, for the award of Doctor of Philosophy.**

**Edinburgh Napier University**

**School of Computing**

**May, 2019**

**Acknowledgement**

It is with deep sense of gratitude that I say thanks to my Director of Studies, Prof. William J. Buchanan and Supervisor, Dr. Gordon Russell for their love, kindness and transformational influence during this research. I also want to thank the entire staff and researchers at Edinburgh Napier University, especially Drs. Lu Fan and Owen Lo, Prof. Ahmed Al-Dubai for their supports that enabled the completion of this work.

I need to say a warmth thank you to my angelic wife, children and extended family for their supports, love, and peace that sustained me although this work.

## Dedication

To Him through whom all things consist, to my angelic wife and lovely children, and more to the fellow saints on Calvary journey.

I cannot but also dedicate this thesis in memory of Prof. David Benyon, who initially was my Panel Chair but could not see the end of it.

## Declaration

I declare that no portion of this work has been submitted for any degree or professional qualification in any university or institution of learning elsewhere.

## Table of Contents

# List of Figures

# List of Tables

# Abstract

Scaling big data infrastructure using multi-cloud environment has led to the demand for highly secure, resilient and reliable data sharing method. Several variants of secret sharing scheme have been proposed but there remains a gap in knowledge on the evaluation of these methods in relation to scalability, resilience and key management as volume of files generated increase and cloud outages persist. In line with these, this thesis presents an evaluation of a method that combines data fragmentation with Shamir’s secret sharing scheme known as Fragmented Secret Share System (FSSS). It applies data fragmentation using a calculated optimum fragment size and encrypts each fragment using a 256-bit AES key length before dispersal to cloudlets, the encryption key is managed using secret sharing methods as used in cryptography.

Four experiments were performed to measure the scalability, resilience and reliability in key management. The first and second experiments evaluated scalability using defined fragment blocks and an optimum fragment size. These fragment types were used to break file of varied sizes into fragments, and then encrypted and dispersed to the cloud, and recovered when required. Both were used in combination of different secret sharing policies for key management. The third experiment tested file recovery during cloud failures, while the fourth experiment focused on efficient key management.

The contributions of this thesis are of two ways: development of evaluation frameworks to measure scalability and resilience of data sharing methods; and the provision of information on relationships between file sizes and share policies combinations. While the first aimed at providing platform to measure scalability from the point of continuous production as file size and volume increase, and resilience as the potential to continue operation despite cloud outages; the second provides experimental frameworks on the effects of file sizes and share policies on overall system performance.

The results of evaluation of FSSS with similar methods showed that the fragmentation method has less overhead costs irrespective of file sizes and the share policy combination. That the inherent challenges in secret sharing scheme can only be solved through alternative means such as combining secret sharing with other data fragmentation method. In all, the system is less of any erasure coding technique, making it difficult to detect corrupt or lost fragment during file recovery.

# Acronyms

| | |
|---|---|
| 3DES | Triple Data Encryption Standard |
| AES | Advance Encryption Standard |
| AONT-RS | All-Or-Nothing-Transform Reed-Solomon |
| API | Application Programme Interface |
| AWS | Amazon Web Services |
| Cloudlets | Cloud Storage Resources |
| CP | Cloud Provider |
| CSA | Cloud Security Alliance |
| CSP | Cloud Service Provider |
| CSS | Computation Secret Sharing |
| DDR | Double Data Rate |
| DES | Data Encryption Standard |
| DR | Disaster Recovery |
| DSA | Digital Signature Algorithm |
| FADE | File Assured Deletion |
| FileID | File Unique Identifier |
| FSSS | Fragmented Secret Sharing Scheme |
| GB | Gigabyte |
| HAIL | High Availability |
| HDD | Hard Disk Drive |
| IBM | International Business Machine |
| IDA | Information Dispersal Algorithm |
| IT | Information Technology |
| KB | Kilobyte |
| MB | Megabyte |
| MD4 | Message Digest 4 |

| | |
|---|---|
| MD5 | Message Digest 5 |
| NIST | National Institute of Standards and Technology |
| NTFS | New Technology File System |
| PDP | Proof of Data Protection |
| POR | Proof of Readability |
| PSS | Perfect Secret Sharing |
| PSSS | Proactive Secret Sharing Scheme |
| RAID | Redundant Array of Independent Disks |
| RAM | Random Access Memory |
| RESCUE | Fragmented Secret Share Project name |
| RSA | Rivest-Shamir-Adleman cryptosystem |
| RTO | Recovery Time Objective |
| SHA | Secure Hash Algorithm |
| SSMS | Secret Sharing Made Short |
| SSS | Shamir Secret Sharing |
| SSSS | Social Secret Sharing Scheme |
| US | United States of America |
| UUID | Universal Unique Identifier |
| VM | Virtual Machine |
| WTSSS | Weighted Threshold Secret Sharing Scheme |

# 1 Introduction

## 1.1 Background

Information Technology (IT) resources available in the cloud have made the adoption attractive in service sectors [1] as a result of growth in knowledge economy placing an emphasis on the provision of consistent data availability for quick decision making. However, with the growth of data regarding types and sizes, concerns have been raised on how best to transmit data securely as well as share and make them readily available uninterruptedly irrespective of the size and type.

Adi Shamir [2] and George Blakely [3] did classic publications in 1979 on how to share data securely without using encryption key known as keyless encryption by defining a method that breaks data (secret) into a number of shares and a certain number of these that can come together to recover the secret of which any number less than these cannot, later to be known as Secret Sharing Scheme. The number of shares created is equivalent to the number of participants, and the number that is required to recover the secret is known as the threshold, this is known as share policy. This scheme uses two main protocols (algorithms) of secret share creation and recovery. The application has proved to be secure, resilient, reliable and efficient in sharing and recovering data in a distributed system. The use of secret sharing scheme has some inherent limitations, of which are the inability to provide data operations at large-scale data size and the effects of changing share policies on system overheads.

Keyless encryption implies breaking data into shares in such a manner that each share of the data exists in a meaningless manner and only a certain defined number known as threshold or more can come together to recover the original data.

A share policy implies a defined threshold ($m$) and maximum number of shares ($n$) to be made from a data. Where data recovery is only possible when the threshold shares ($m$) or number of shares equal to the total number of shares ($n$) are put together using an algorithm.

The word cloudlets, Cloud Service Providers (CSPs) and clouds are used interchangeably in this thesis.

### 1.1.1 Cloud systems

In the face of these current realities, we present an evaluation of a method for sharing large-scale data infrastructure in multi-clouds using a combination of data fragmentation and secret share scheme. A system that can provide consistent data availability, high-level scalability and security, as well as maintaining data integrity within cloud-based architecture know as Fragmented Secret Share System (FSSS). It creates fragments from a file, encrypts each fragment and applies secret sharing methods as used in cryptography to create robust and secure keyless key management system in a multi-clouds data distribution system.

The process involves the user providing the file(s) as well as choosing desired share policy for each operation, while the system provides appropriate optimum fragment size and number of cloudlets that will participate in the operation. It goes forward by breaking the file into chunks using the chosen fragment size, encrypting each chunk with different AES-256-bit key generated by a random key generator and then creates shares out of the encryption key based on user's chosen share policy. The shares, as well as the encrypted fragments are stored with selected CSPs, and when the file is required, the key shares are recovered using the same key share policy in relations to the defined threshold and as well as the encrypted fragments. Each recovered key is therefore used to decrypt corresponding encrypted fragment, and with the fragments decrypted serially, the original file is recombined and file checksum performed using SHA512 to ensure file integrity before delivery to the file owner.

### 1.1.2 Secret shares

Shor *et al.* [4] suggests that the optimal way of sharing big data in multi-cloud environments is in the combination of data encryption and use of efficient secret sharing scheme in managing encryption key in the light of this, four experiments were performed to measure scalability, resilience and key management of FSSS and evaluate same with the evaluation frameworks developed by this thesis on the areas of scalability and resilience. The first experiments used defined fragment blocks to break file of varied block sizes - 1KB, 10KB, 100KB, 1MB, 10MB, 100MB and 1GB into fragments. The second experiments used an optimum fragment size defined as 15% of file size (this was derived from the observation of result trends in our previous work [5]) and used in breaking the above file sizes into fragments. Both were used in a combination of different secret sharing policies and our results showed that defining an optimum fragment size of 15% of file size produced less overhead than the first experiment. The second experiment also provided high-level scalability at different secret sharing policies than the first. In the third experiments, file recovery during cloud failures were tested and showed that recovery were faster in as much as the outage does not exceed the defined threshold of the secret sharing policy in use. While the fourth experiment showed that improving on the concept of socialisation in secret sharing scheme as earlier proposed by Nojoumian *et al.* in 2010 could provide good key management system in multi-cloud architecture that can mitigate losses occasioned by cloud outages.

The results of evaluation of FSSS with respect to similar methods using the frameworks designed by this thesis suggest that the method of using calculated optimum fragments size in breaking data into fragments and managing encryption keys with an efficient secret sharing scheme provides much more optimal means of sharing large-scale data infrastructure than suggested by *Shor et al.* [4]. In addition, it also provided a *4-in-1* level of security to data through fragmentation, encryption, redundancies and robust key management through secret share system. With the resilient nature of the system even in the face of 60% cloud failures as well as the robust key management system, the method showed promises in redefining cloud-based disaster management from that of system recovery to mitigation of losses occasioned by cloud-based disasters. In all, the system is weak in the area of detecting malicious, corrupt or lost fragment during file recovery and does not implement a threshold scheme nor applied an erasure coding technique in data storage management.

## 1.2 Disaster Recovery and Fail-Over System

An estimate that one in every four businesses will not be able to survive a disaster according to US government makes information technology disaster recovery plan an invaluable investment for business owners [6]. Disaster, as an unexpected event in a system lifetime, can be natural or man-made and its recovery could be traditional or cloud-based [7]. In all, the essence of disaster recovery system is for business continuity, and this needs to be pursued within the least possible cost to business owners during failover or failback phase.

OnlineTech [8], in their disaster recovery white paper listed in order of priorities, increased reliance on technology, increased business complexity, increasing frequency and intensity of natural disasters, increased reliance on third-parties and others as reasons why business owners should consider disaster recovery plan as a way of ensuring business continuity. In all, this thesis tends to change the status quo from disaster recovery to disaster mitigation through robustness and resilience using data fragmentation technology, secret sharing scheme as well as multi-cloud architecture. Figures 1 and 2 show an example of an existing model as designed by Gu *et al.* in [9]. In Figure 2, Cloud Provider 1 ($CP_{1)}$ is the major data disaster recovery provider to customers of different levels, while in Figure 1, $CP_2$, $CP_3$ to $CP_M$ are collaborative cloud service providers that provide other cloud resources to support $CP_1$ in Figure 2.

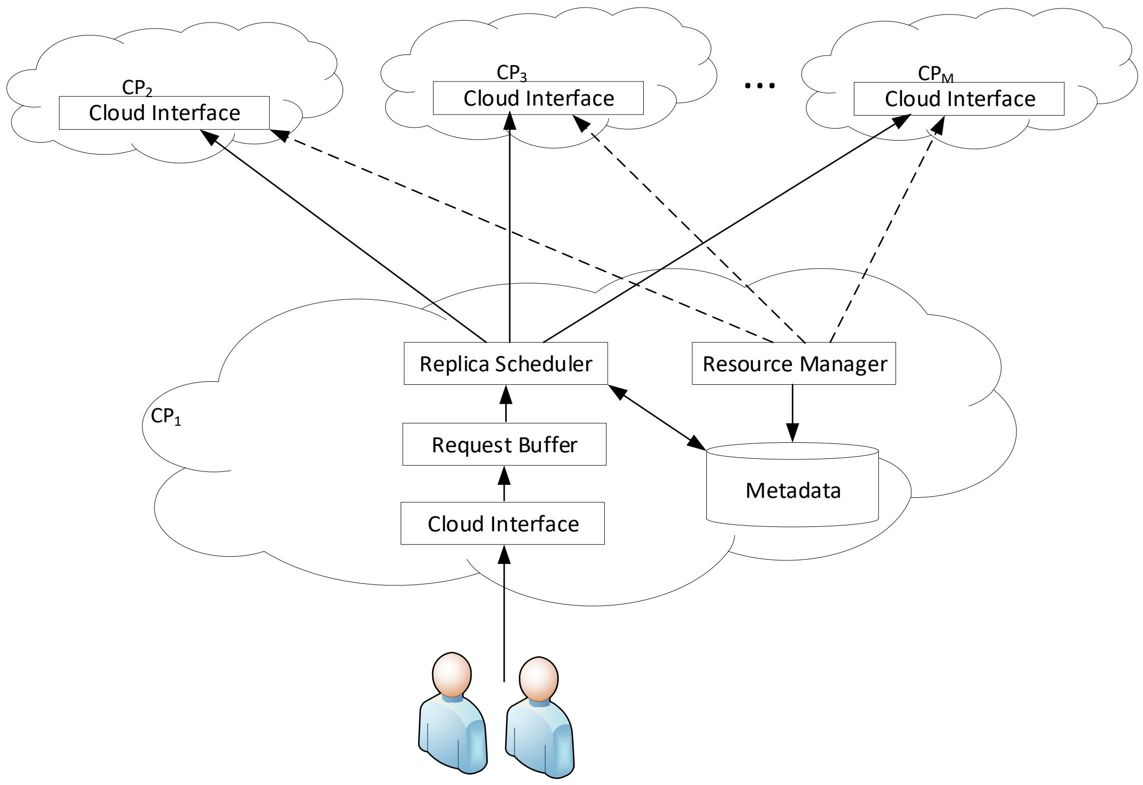


Figure 1: Data Backup Model [9]

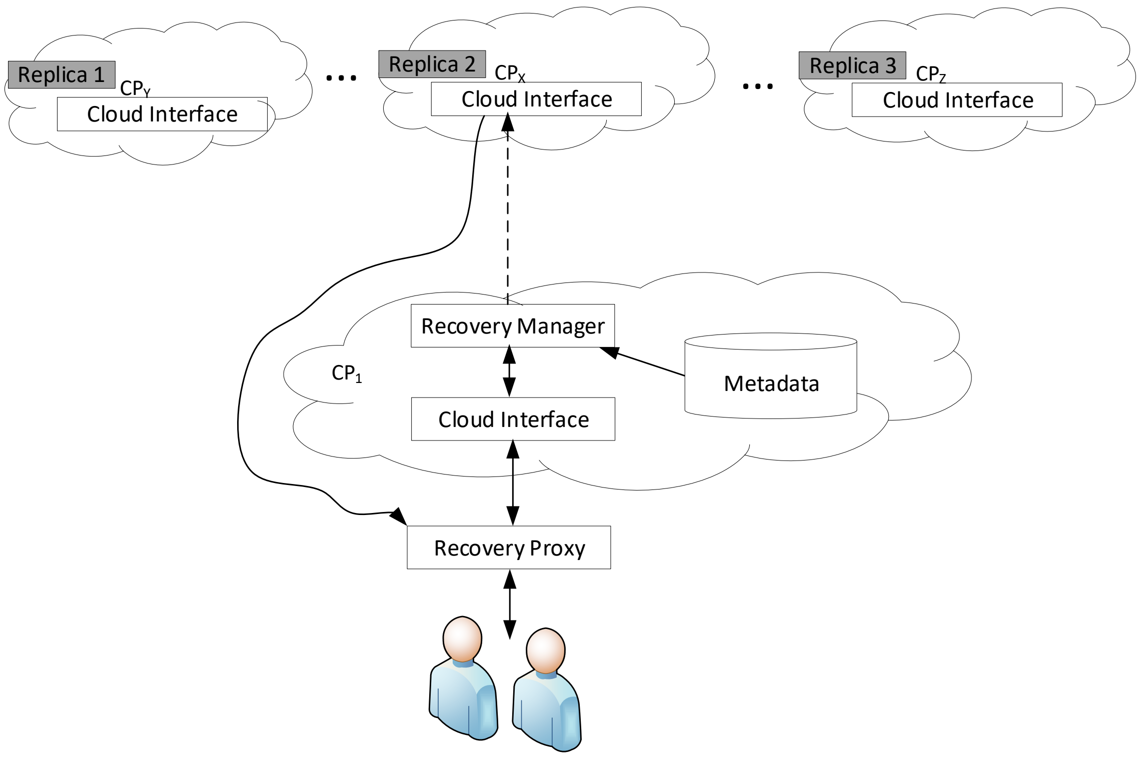


Figure 2: Data Recovery Model [9]

## 1.3 Secret Sharing Scheme

The deviation from key-based to keyless encryption was introduced by Adi Shamir and George Blakely in 1979 [2, 3] in two different seminal papers, each presented to the world a different means of securing cryptographic keys. Their works focused on splitting the key into meaningless shares in such a way that it will take only a certain number of the broken keys (shares) known as a threshold to come together and reconstruct the key and any number less than the threshold cannot. This concept was later known to be Secret Sharing Scheme.

This scheme focuses on the techniques used in striping and distribution of data among many participants in such a way that a certain number of the participants, known as the threshold, can come together and recover the original data while a certain number less than the threshold cannot [10]. The Shamir Secret sharing scheme is an ideal scheme and known as a perfect scheme [2], while many provide computational security such as [11]. Krawczyk [11] is of the opinion that an $(n, m)$ −secret sharing scheme is a randomised protocol that stripes a secret $S$ and disseminate same as shares to an $n$ participants in such a way that only an $m$ participants shares are capable of recovering the original secret for $m$, $1 \leq m \leq n$, while $m - 1$ shares cannot give any information on the secret as presented by [2, 3].

Using real-life scenario with diagrammatic representations in figures *3* and *4*, we take for instance a beverage company, the owner wants to make their recipe a top secret. Their intention is to prevent their managers from learning the recipe and in so doing decided to use an algorithm to break their recipe into shares ($S_i$) in such a way that a certain number of the shares ($M$) will be enough to recover the recipe out of the total number distributed to managers ($N$). After breaking the recipe into shares, the shares were distributed to say four of their top managers knowing that at least two from the four are needed to get the recipe back. This total number of shares made of the recipe is equivalent to the number of players (Managers) and the minimum number required to get the recipe back is known as a threshold.

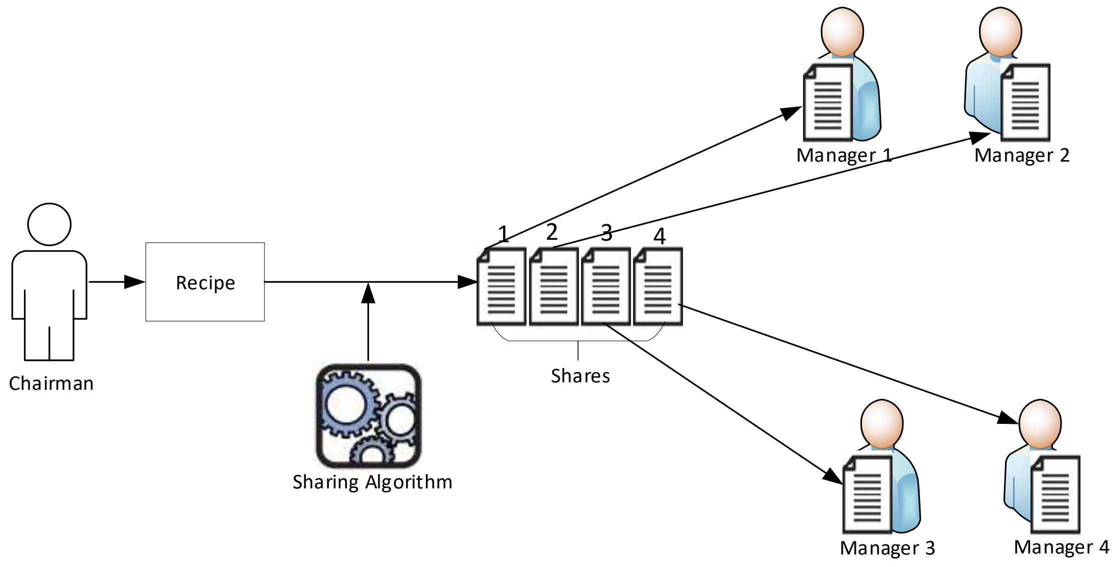


Figure 3: Diagram of a Secret Sharing System

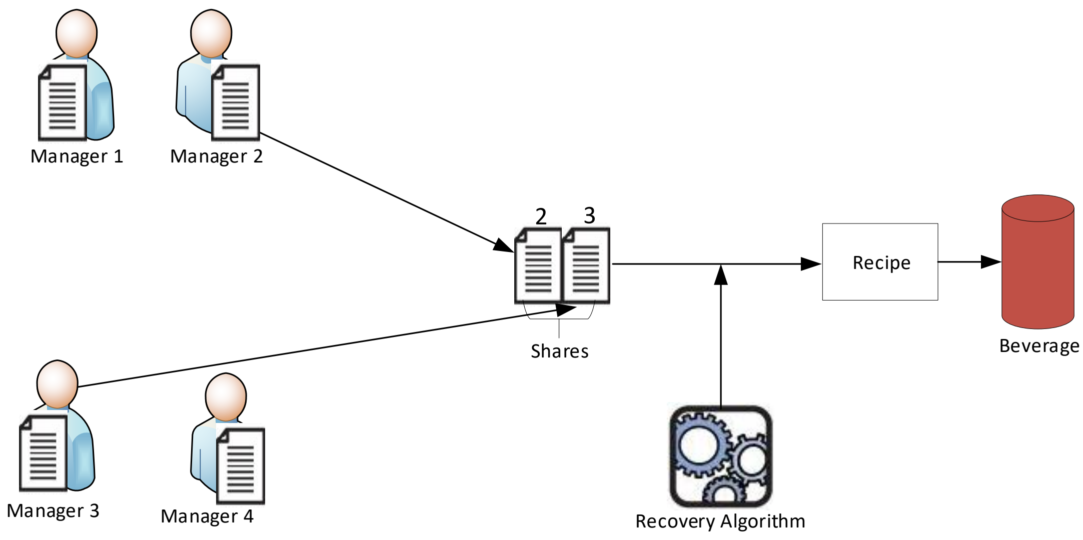


Figure 4: Diagram of a Secret Recovering System

## 1.4 Research Questions

The core research questions which this thesis aims to address are:

- Several methods of data fragmentation in conjunction with secret sharing scheme have been proposed as fit for use in multi-cloud architecture, but with the advent of big data infrastructure, which of these methods can continue operation at different changes of file size and share policy making it suitable for use in large-scale data infrastructure?
- The use of multi-cloud architecture is to improve on redundancy technique and make data readily available. As cloud failures persist, is there a storage method that can ensure consistent data availability irrespective of cloud failures, and how does one understand different rates of cloud failure based on chosen share policy?
- Data fragmentation technique that combines secret sharing scheme for key management has been adjudged better in performance in multi-cloud environment. How does this method tend to overcome the complexity of key retrievals during adverse situations as key retrievals are impossible when outage exceeds defined threshold?

## 1.5 Aim and Objectives

This thesis is focused on the evaluation of fragmented secret share system, a method for sharing large-scale data infrastructure in multi-clouds using a combination of data fragmentation and secret share system with respect to scalability, resilience and key management in relation to current available practices. To be able to meet the above stated aim, these objectives are therefore defined:

**Objective 1: To build an evaluation framework that defines scalability using the metric of continuous operation at different changes of file size and share policy.**

Different methods of data fragmentation exist in data sharing and retrieval in multi-cloud environment but there is a gap in knowledge on what constitutes scalability of these methods and how to evaluate them. The interest of this thesis is on such methods that incorporate secret sharing scheme in its data fragmentation technique. In view of this, an evaluation framework that defines scalability in such a manner that when file size and share policy change, there is no significant effect on data sharing and retrieval is developed. This thesis also uses same to evaluate different data fragmentation methods in multi-cloud systems in order to assess their suitability for use in large-scale data infrastructure.

**Objective 2: To further develop an evaluation framework that defines resilience of a method as the ability to avoid downtime and data losses during acceptable level of cloud failures.**

Use of secret sharing scheme in data sharing in multi-cloud architecture are of two options, data sharing and key management. These explore the resilient nature of secret sharing scheme in their implementations, measuring this resilient nature using an evaluation framework has therefore become very important. This thesis using the existing knowledge in literature that the acceptable level of cloud outage in these methods lies only when the number of shares to recover original secret is greater than or equal to the threshold ($n \geq m$). In view of the above, this thesis therefore evaluates different methods' ability to validate this fact, the relevant information about the behaviours of CSPs at each rate of cloud outage and what is being done to mitigate the above bottleneck.

**Objective 3: To evaluate how different methods used in data sharing in multi-cloud architecture handle the perceived complexity in key management.**

It is believed that this complexity is more on retrieval process as the more complex the retrieval interface is, the greater the challenge of deploying key retrieval in applications such as decrypting encrypted file. The evaluation of how different methods of key management fits into the above definition of resilience in Objective 2, and the implication of its application in order to provide a hitch-free key retrieval process at different rates of cloud outage in addition to information on what is being done through this thesis to propose a mitigating factor using self-organisation forms part of our objective. Furthermore, providing information on the potential of fragmented secret sharing scheme in mitigating losses occasioned by cloud-based disaster rather than minimising the recovery time objective of cloud-based disaster remains one of this focus.

## 1.6 Contributions

- Cloud storage resources provide quick access to data as computing is usually on the go. Using multi-cloud architecture for storage therefore increases access level as the system can rely on more than one cloud storage resources [9], [12], [13], [14], [15], and [16]. Secret sharing scheme as a method of data sharing in multi-cloud has remain a focus of research in this direction as many variants exist but the argument of its suitability for sharing large-scale data infrastructure persists. Alsolami and Boult in [17] are of the opinion that the use of Secret Sharing is an ideal solution and this position is supported by the works of Ermakova and Fabian [13] and Fabian *et al.* [12] but in all failed to scale data size above 500MB. FSSS by using a defined optimum fragment size (within the scope of this thesis experiments) and secret sharing in key management was able to continue production as file size increased exponentially making it suitable for use in large-scale data infrastructre.

- Shor *et al.* [4] by their results suggests that the optimal way of sharing big data in multi-cloud environments is in the combination of data encryption and use of efficient secret sharing scheme in managing encryption key. But in the contrary, FSSS found out that using its defined optimum fragment size in breaking the data into fragments before encryption and managing encryption keys using efficient secret sharing scheme provide an optimal and highly scalable scheme that is best fit for use in large-scale data infrastructure.

- This thesis evaluation shows two main methods for data sharing in cloud-based systems. While one method combines data fragmentation with secret sharing scheme for key management of encryption keys such as Kapusta *et al.* [18], Abdallah and Salleh [19], Koikara *et al.* [20], and Pal *et al.* [21]. Another method relies only on secret sharing scheme for data fragmentation, like the works of Alsolami and Boult [17], Ermakova and Fabian [13] and Fabian *et al.* [12] but none provided information on the relationship between file sizes and share policies at different applications scenario and this FSSS provides using extensive experimental frameworks.

- There are many methods of data sharing in multi-cloud architecture, but the interest of this thesis is limited to data sharing methods that utilises secret sharing scheme in its design methodology. In order to provide a secure content sharing of data in public cloud, Xiong *et al.* [22] through CloudSeal provided a system of data sharing that integrates multi-cryptographic system such as symmetric encryption, proxy-based re-encryption, and threshold, *m-out-of-n* secret sharing scheme in conjunction with broadcast revocation mechanisms and by so doing provide an end-to-end security and privacy of distributed data in pubic cloud. Fabian *et al.* [12] in other hand developed a method of collaboratively and inter-organisational sharing of big data in healthcare sector using attributed-based encryption for access control and secret sharing scheme for data sharing in multi-clouds. DEPSKY [23], a system that improves the availability, integrity and confidentiality of information stored in the cloud through the encryption, encoding and replication of the data on diverse clouds that form a cloud-of-clouds is a good example of real life application of Secret Sharing scheme in multi-cloud-based storage system. Other works like CloudStash [24] utilises secret-sharing, low cost cloud storages and multi-threading to improve confidentiality, availability, performance and fault tolerance. Works like ARCHISTAR [25] by Loruenser *et al.*, Li *et al.* in CDStore [26], and so on are some of the examples of data sharing methods that utilise secret sharing in multi-cloud architecture. From the above, there is no known work that contributed on the evaluation of these methods of data sharing with the view of finding their strengths and weaknesses to establish their suitability of applications in sharing large-scale data infrastructure using multi-cloud storage facilities. Providing this critical information forms part of the contribution of this thesis.

- Some of the works above extensively utilised secret sharing scheme in data sharing, thereby providing keyless encryption in securing data such as that of Fabian *et al.* [12], Loruenser *et al.* [25] in ARCHISTAR, Ermakova and

Fabian [13] in defining a method that used secret sharing for health data in multi-provider clouds and Alsolami and Boult [24] in CloudStash: Using Secret-Sharing Scheme to Secure Data, Not Keys, in Multi-Clouds. In all, none of these works established relationships between file size and share policy combinations at both normal and adverse application scenarios such as cloud outages.

- Use of secret sharing scheme has some inherent limitations, of which are the inability to provide data operations at large-scale data size and the effects of changing share policies on system overheads. The works of Abdallah and Salleh in [19] and [27] provided extensive information on all these without any known solutions and these are what FSSS provided using fragmented secret share system through defined optimum fragment size. By developing two evaluation frameworks on scalability and resilience, this thesis defined scalability as the ability to continue production even when file size increases exponentially and resilience of secret sharing as the ability to continue production in multi-cloud environments during adverse cloud failures. These FSSS's results and future works suggest are the better approach than existing methods.

## 1.7 Thesis Structure

This thesis is structured as:

- **Chapter 1. Introduction** – Introduces some basic concepts of the work that will help capture the essence and knowledge gap being investigated as well as the proposed solutions. It has in details the research question, aim and objectives, comprehensive research overview and expected results from the research.
- **Chapter 2. Background** – This chapter introduces the fundamentals of computer security alongside the uses and application of data striping technique, which forms the background technology for data dispersal in this report and FSSS architecture.
- **Chapter 3. Literature Reviews** – Secret Sharing Schemes and their basic classification are discussed alongside its security limitations as there is no perfect security in cryptography. Because cloud storage forms major components of this work, a thorough and concise review of relevant literature relating to security of

the stored data in the cloud is also presented alongside that of disaster recovery as well.

- **Chapter 4. Experimental Design** – This chapter is concerned with the steps used in the experimental setup through coding, execution of codes, result collections and computations to determine cloud behaviours during process executions.
- **Chapter 5. Data Fragmentation Evaluation in a Multi-Cloud Environment** – This thesis used this chapter to evaluate a method of fragmentation known as Fragmented Secret Share System (FSSS) alongside similar schemes that are built on Secret Sharing scheme so as to measure their levels of scalability in a multi-cloud environment.
- **Chapter 6. Key Management Evaluation in a Multi-Cloud Environment** – In this chapter, FSSS key management is presented with a view of evaluating same with similar methods that use secret sharing scheme in key management in the cloud as FSSS combines data fragmentation with encryption and manages the keys using Shamir's secret sharing scheme.
- **Chapter 7. Disaster Management Evaluation in a Multi-Cloud Environment** – This chapter aims to present some features of FSSS in comparison with similar methods that make it reliable, readily available and resilient, and how it tends to use these qualities to redefine disaster management using cloud-based resources from that of recovery from losses to mitigation against losses.
- **Chapter 8**. **Conclusion and Future Works –** This chapter presents the thesis summary, major findings, limitations and Future Works.

## 1.8 Publications

The publications that are core to this thesis are:

- Buchanan, W. J., Lanc, D., **Ukwandu, E**., Fan, L., Russell, G., & Lo, O. (2015). The future internet: A world of secret shares. *Future Internet*, *7*(4), 445-464.
- **Ukwandu, E**., Buchanan, W. J., Fan, L., Russell, G., & Lo, O. (2015, August). RESCUE: Resilient Secret Sharing Cloud-Based Architecture. In Trustcom/BigDataSE/ISPA, 2015 IEEE (Vol. 1, pp. 872-879). IEEE.
- **Ukwandu, E.,** Buchanan, W. J., & Russell, G. (2017, June). Performance evaluation of a fragmented secret share system. In *Cyber Situational Awareness,*

*Data Analytics And Assessment (Cyber SA), 2017 International Conference On* (pp. 1-6). IEEE.

- Buchanan, W. J., **Ukwandu, E.,** van Deursen, N., Fan, L., Russell, G., Lo, O. & Thuemmler, C. (2016). Secret shares to protect health records in cloud-based Infrastructures. In 2015 17th International Conference on E-health Networking, Application & Services (HealthCom). Doi:10.1109/HealthCom.2015.7454589. ISBN 978-1-4673-8325-7.

# 2 Background

## 2.1 Introduction

This chapter provides the fundamental mathematical backgrounds and theory that the thesis will be based on. Two major concepts that feature prominently in this study – Data sharing and Cloud-based storage, as well as the supporting information theory has been brought to fore. Moreso, providing insight into some inter-disciplinary units that make up our research has become necessary here to a foster better understanding of this thesis.

## 2.2 Fundamentals of Computer Security

The human information need and the quest to meet with the demand has brought with it a new stage of the revolution known as a knowledge economy. In the face of these realities lie the fear of how secured the information being transmitted from one computer to another; the authenticity of the message signature and the assurance that the information being transmitted still retains its original contents. In dealing with the above subject matters, computer scientists through the years have been working on the science of disguising a message in such a way to hide its substance. This entails disguising a message (plaintext) into scrambled text (ciphertext), in a process called encryption. The process of turning the ciphertext back to its original format is called decryption.

The art and science of keeping message secure is known as cryptography [24]. Encryption and decryption involve the use of keys to be able to encrypt and decrypt messages without fail, and these keys are usually exchanged between the parties involved in the process. This can be private key (Symmetric), private and public key (Asymmetric) or one-way hash method (Hash function). In transmitting these keys from one computer to another computer scientist use different mathematical algorithms. In symmetric key algorithms, we have DES, 3DES, AES, RC4, Twofish and so on, while Asymmetric Key we also have RSA, DSA, ElGamal and so on. In one-way hashing, we have examples like MD4, MD5, SHA-1, SHA-256, and SHA-512.

## 2.3 Data Striping: Uses and Applications

In a bid to bring solution to key management problems associated with key-based encryption as discussed above, a keyless encryption system was introduced with great impact in storage technology and one method to achieve this is data striping. This method is successfully applied in a distributed storage system. In storage technology, data striping is used to break data into chunks and subsequently dispersed to multiple storage disks. Striping technique as applied to several disks comes to bear when a data processing device requests data more frequently than a single storage device can handle. Its application is spread across the accessing of concurrent multiple devices as it increases total data throughput.

One of its usefulness factors is in balancing input/output load across an array of disks such as in Redundant Array of Independent Disks (RAID). In FSSS as shown in Figure 5, we used data fragmentation, encryption and secret sharing scheme. The three as above are used in securing and dispersing data to different cloudlets as well as the encryption keys. A Splitter first breaks a file into fragments using a predefined optimum fragment size, calculated basically in relation to a percentage of the file for which most often lies from 15% of file size as derived from observations and measurements from previous results in [5]. The fragments each are encrypted with an AES-256 encryption key randomly generated by a key generator. Each of these keys is stored using Shamir's secret sharing scheme, where shares generated are dispersed to multiple cloudlets for which a predefined threshold is used to recover the key when needed. To recreate the file, the shares are recovered from the cloudlets for each key using recovery algorithm and uses so to decrypt each fragment and thereafter recreate the file by bringing the decrypted fragments together using a Combiner.

## 2.4 Mathematical Background

### 2.4.1 Information Theory

This work centres on data distribution and its security within a cloud infrastructure and therefore will look into the mathematical background of information security and hence Information Theory. Claude Shannon and Warren Weaver [29] published in 1948 a classic seminal paper that introduced the modern and meaningful way of thinking about communication of information. Information theory studies the transmission, processing, utilisation, and extraction of information. It defines the amount of information contained in a message as the minimum number of bits needed to encode all possible meanings of the message, in as much as they are equally likely. In simple terms, information can be thought of as the resolution of uncertainty.

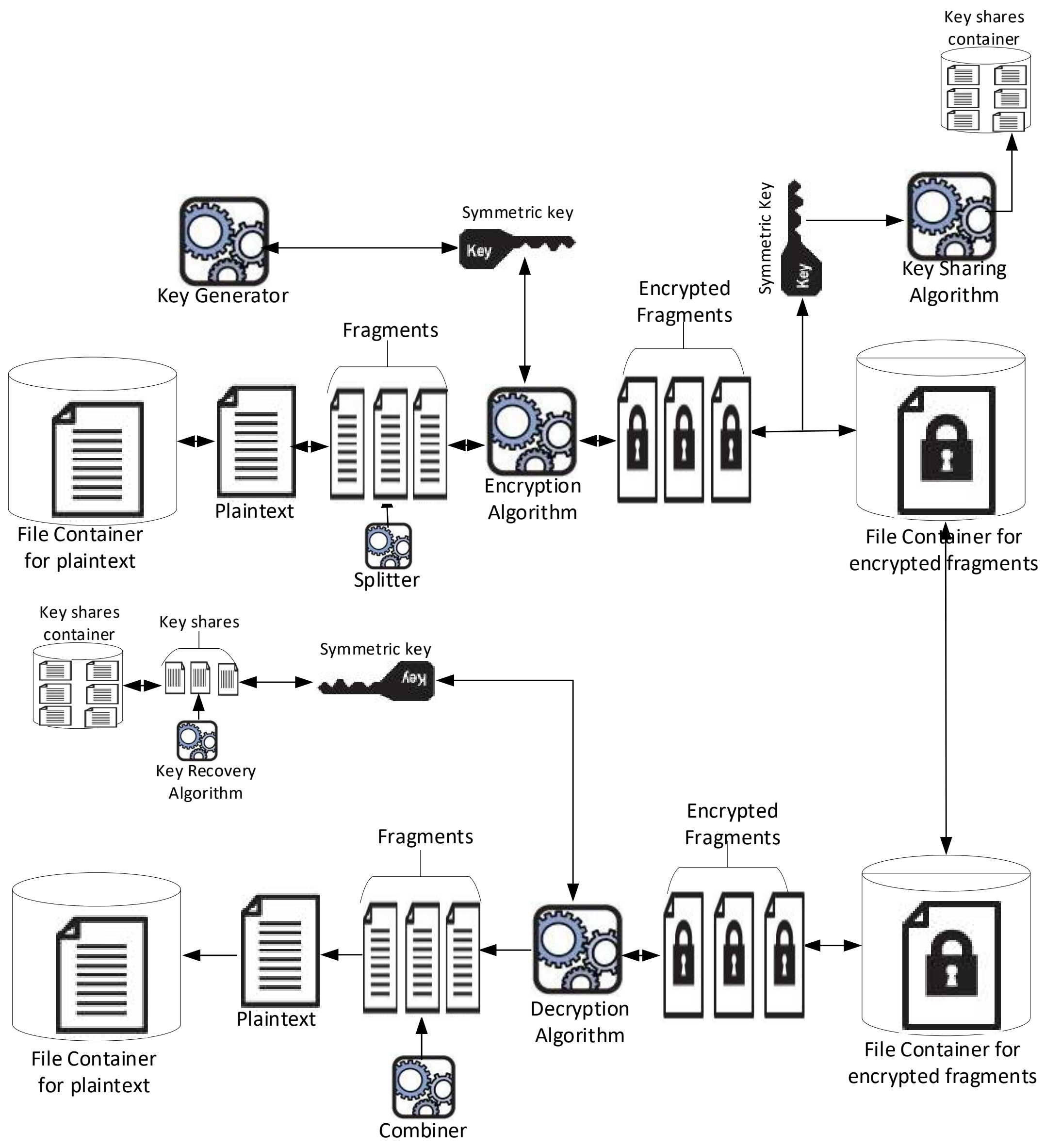


Figure 5: Data Fragmentation and File Reconstruction

## 2.4.2 Definition and History

Schneier [30] defines entropy measures as the amount of information or uncertainty contained in a message *M* given as *H(M)*. Shannon defined the actual rate of a language mathematically as [30]:

$$r = \frac{H(M)}{N} \qquad \text{Equation 1}$$

Where *H(M)* is the entropy, *N* is the length of the message and *r* is the rate of the language. The absolute rate, which is the maximum bits size that can be coded in a character, assuming all characters are equally probable, is given as [30]:

$R = log_2 L$ Equation 2

$L$ is the number of characters in a language; the equation defines the maximum entropy of the individual characters. The redundancy of a language $D$ is therefore given as [30]:

$D = R - r$ Equation 3

The information content of a symbol or event is defined by its probability. In relating this work to the security of a cryptosystem, Shannon theorised that there is nothing like perfect secrecy as ciphertext to the barest minimum reveals information about the plaintext, this it does by not being able to have a key as long as the plaintext (message) without possibly reusing any symbol, or character so as to be able to achieve perfect secrecy. Perfect secrecy is only possible in a One-Time-Pad. The entropy of a cryptosystem is therefore a measure of the key size K, given as [30]:

$H(K) = log_2 K$ Equation 4

It is therefore worthy to note that the larger the entropy of a cryptosystem, the more difficult it is to crack. As information refers to the degree of uncertainty contained in a situation, it therefore correlates that the larger the uncertainty removed by a message, the stronger the correlation between the input and output of a communication channel.

### 2.4.3 Perfect Secret Sharing Scheme

Shamir (1979) work provides a good example of perfect secrecy. The scheme, which is a keyless scheme is known as perfect secret sharing [2]. It involves two protocols, secret sharing and recovering. Shamir's Perfect Secret Sharing (PSS) relies on the idea that on the principle that you can define a straight line with two points, three points for a quadratic equation, and so on, to give $t$ points to define a polynomial of degree $t-1$. Hence, a method for $t$-out-of-$n$ secret sharing can thus use a polynomial with a $t-1$ degree using a secret for the first coefficient, and then random values for the remaining coefficients. Next, find $n$ points on the curve and give one to each of the players. As a result, when at least $t$ out of the $n$ players reveal their points, there is sufficient information to fit a ($t-1$)th degree polynomial to them, in which the first coefficient is the secret. There are two conditions required for it to be perfect: if, and only if, $t-1$ shares provide absolutely no information regarding the hidden secret, and when the ratio of the length of the secret to the length of each of the shares (known as the information rate) is equivalent to unity.

To illustrate this construction technique, consider a concrete example. Assume that a secret value is split into five parts, three of which are needed to reconstruct the original data (*i.e.*, $n = 5$, $t = 3$). The first step is to create a second-order polynomial [31]:

$$y = a_0 \times x^0 + a_1 \times x^1 + a_2 \times x^2 \qquad \text{Equation 5}$$

The second step is to assign the secret value to the first coefficient (*i.e.*, $a_0$) and choose random values for the remaining coefficients (*i.e.*, $a_1$ and $a_2$). Suppose that the secret value is: 42, $a_1 = 1$ and $a_2 = 2$, then the polynomial becomes [31]:

$$y = 42 + x + 2 \times x^2 \qquad \text{Equation 6}$$

The third step is to calculate any five ($x$, $y$) pairs, for instance:

| x | | y |
|---|---|---|
| 1 | $42 + 1 + 2 \times 1$ | 45 |
| 2 | $42 + 2 + 2 \times 4$ | 52 |
| 3 | $42 + 3 + 2 \times 9$ | 63 |
| 4 | $42 + 4 + 2 \times 16$ | 78 |
| 5 | $42 + 5 + 2 \times 25$ | 97 |

Finally, distribute one pair to each player, e.g., Player1 receives (1, 45), Player2 receives (2, 52), and so on. No player can thus tell anything about the original secret, unless at least three players exchange their information and yield equations like:

$$45 = a_0 + a_1 \times 1 + a_2 \times 1^2$$

$$52 = a_0 + a_1 \times 2 + a_2 \times 2^2$$

$$63 = a_0 + a_1 \times 3 + a_2 \times 3^2$$

Hence, the three players together can work out that the secret value $a_0$ is 42. This method works fine, but from a security point of view, a player can still obtain more information about the secret with every pair on the polynomial that they find. For example, player Eve finds two pairs (2, 52) and (4, 78). Although these are not enough to reveal the secret value, Eve could combine them together and get:

$$76 = a_0 + a_1 \times 4 + a_2 \times 4^2$$

$$52 = a_0 + a_1 \times 2 + a_2 \times 2^2$$

Therefore, Eve can work out that:

$a_0 = 26 + 8 \times a_2$

So, Eve starts to replace $a_2$ with 0, 1, 2, 3… to find all the possible values of $a_0$. This problem can be fixed by using finite field arithmetic in a field of size *r* where:

$r > a_i$, $r > N$, $r = p^k$, $p \in$ P where P is the set of primes and *k* is a positive integer.

Then, calculate the pairs as [31]:

$$y = f(x) \ (mod\ p) \qquad \text{Equation 7}$$

Where $f(x) = a_0 \times x^0 + a_1 \times x^1 + a_2 \times x^2$, as in equation 5 above. For example, consider the example of $p = 61$:

| *x* | *p* | | *y* |
|---|---|---|---|
| 1 | 61 | (42 + 1 + 2 × 1) % 61 | 45 |
| 2 | 61 | (42 + 2 + 2 × 4) % 61 | 52 |
| 3 | 61 | (42 + 3 + 2 × 9) % 61 | 2 |
| 4 | 61 | (42 + 4 + 2 × 16) % 61 | 17 |
| 5 | 61 | (42 + 5 + 2 × 25) % 61 | 36 |

Shamir's PSS is a promising approach to secret sharing for cloud storage, and provides many advantages, including:

1. **Secure**—Anyone with fewer than *t* shares has no extra information about the secret than someone with zero shares.
2. **Extensible**—When *n* is fixed; new shares can be dynamically added or deleted without affecting the existing shares.
3. **Dynamic**—With this it is possible to modify the polynomial and construct new shares without changing the secret.
4. **Flexible**—In organisations where hierarchy is important, it is possible to supply each of the participants a different number of shares according to their importance.

### 2.4.4 The Social Concept in Secret Sharing

The social concept in secret sharing as proposed by Nojoumian *et al.* [4, 5, 6] suggests that the resilient nature of secret sharing scheme can be strengthened by using it to develop a self-organising system as it concerns the use of cloud storage resources. The concept mimics human social interactions where participants during social tuning phase are allocated more shares or disenrolled based on their reputation using calculated trust function that is based on response time and rate of availability during sharing and recovering phases. This implies that more cooperative players get more shares without exceeding the established threshold, while the non-cooperative ones get disenrolled from subsequent operations.

This concept can be likened to a situation such as in Kit Kat Beverage Company where the Chief Executive Officer wants to make the recipe for their products secured in such a way that none of the top management staff can have it. In doing so, he/she decided to break the recipe into shares. Let us assume the Company has five top management staff named Alice, Bob, Charlie, David, and Grace. He will break the recipe into five shares in such a way that it will take only three out of the five staff to collaborate their shares to have the recipe and any number less than that cannot. Let us represent this mathematically *(Recipe = S, participants (n) = Alice, Bob, Charlie, David, and Grace, Weight of each participants (W)):*

$$S = \sum_{i=0,\mathrm{i}\neq \mathrm{j}}^{n-1} W_j X_i \qquad \text{Equation 8}$$

Where $i$ is the index number of participants, $0 \leq i \leq n-1,\ 0 \leq j \leq m-1$, $m$ is the threshold. At first, the weight ($W$) of all participants are initialised to *1*.

For a recovery process, each participant's weight is determined by the Throughput and Availability Rate. To recover the secret, only the participants with better reputation (faster Throughput and Availability) but not more than the required threshold will be selected:

$$S = \sum_{i=0,\mathrm{i}\neq \mathrm{j}}^{m-1} W_j X_i \qquad \text{Equation 9}$$

## 2.4.5 Non-Perfect Secret Sharing Scheme

In a situation where partial information about the secret has been gained by the $t-1$ subsets of participants that cannot recover the secret value, a scheme known as non-perfect secret sharing scheme has been established [35]. In this scheme, the size of shares *($S_i$)* most often is less than that of the secret *(K)* and therefore produces an information rate less than *1*, as mathematically expressed in [35].

$$f(0) = K \qquad \text{Equation 10}$$

and:

$$\text{Information rate} = \frac{log_2 K}{log_2 S_i} \neq 1 \qquad \text{Equation 11}$$

Where *f(0)* implies that the secret *(K)* lies at the position where *X=0*, which is the intercept, $log_2K$ is the size of the secret and $log_2S_i$ is that of share, and $0 \leq i \leq n-1$.

In Asmuth *et al.* [36] a method to safeguard key was proposed in which the shares are congruent classes of a number associated with the original key. It is an *(m, n)*-threshold secret sharing scheme that applies the use of Chinese remainder theorem as well as prime numbers. As an example, in an $(m,n)$ −threshold scheme, consider a large prime, $p$, greater than *M*. Then choose *n* numbers less than $p, d_1, d_2, \ldots, d_n$,

such that:

1. The *d* values are in increasing order; $d_i < d_{i+1}$
2. Each $d_i$ is relatively prime to every other $d_i$
3. $d_1 \times d_2 \times \ldots \times d_m > p \times d_{n-m+2} \times d_{n-m+3} \times \ldots \times d_n$

The strengths of this scheme is on its efficiency in the sharing and recovery of secret; its sensitivity to random errors; and the relation between the threshold ($m$), the *m-1* that cannot recover the secret ($s$), and the total number of participants ($n$).

Brickell [37] worked on an ideal situation but for more general access structures, dealing with assigning each participant a level, which is a positive integer thereby forming a multilevel access structure consisting of the subsets.

Implying that, at a level of 2, 1 or 2 participants are empowered to determine the secret, whereas, in a level of 3, 2 or 3 participants can. Simmons [38], proposed a compartmented access structure differing from that of a threshold access structure with a more general view. The access structure is arranged in different compartments, which are a disjoint set of participants, say $C_1..,C_u$, containing positive integers $t_l, \ldots t_u$, and *t*. This consists of all subsets containing at least $t_i$ participants from $C_i$ $for$ $1 \leq i \leq u$, and a total of at least *t* participants.

Some non-perfect secret sharing schemes like Information Dispersal Algorithm is discussed in detail in Section 3.22 as their contributions are necessary in understanding the scheme under review.

## 2.5 Conclusions

Information theory as propounded by Shannon and Weaver [29] play an important role in understanding the concept of this thesis as it is centred on data at every stage – rest, motion and in use. So, the understanding of its core principle, background and practical application is important for clarity and better comprehension of this work. Knowing this will foster a better understanding of the mathematics involved in Secret Sharing Scheme as done by Shamir [2] and Blakely [3] in 1979.

# 3 Literature Review

## 3.1 Introduction

This chapter presents an examination carried out to provide insights into the areas FSSS is premised. Relevant literatures are reviewed around the Secret Sharing scheme, such as that of Shamir, Rabin IDA, Krawczyk and Social Secret sharing scheme. The essence is to understand their potentials, weaknesses and applications in data security with reference to cloud data storage security. This thesis pays much attention to the limitations that make such scheme more suitable in key management than data sharing and dispersal in a multi-cloud architecture.

The work of Shor *et al.* [4] is reviewed so as to elicit the reason behind our claim that using optimum fragment size in fragmenting data before encrypting and dispersal rather than encrypting the whole data is more optimal, scalable and suitable for use in large-scale data infrastructure. A review on cloud-based data storage system will help provide knowledge on the characteristics that encourage adoption, some major concerns and steps through research to address these challenges. One major area is the migration from single to multi-cloud storage as a measure to improve on availability and resilience of cloud data storages through redundancy technique. Furthermore, this thesis explored more works that ensure data availability autonomously or otherwise regardless of hardware failures, corrupted physical disks or downtime as well as their several key management methods.

## 3.2 Secret Sharing Scheme

In storage technology, data sharing is used to break data into shares and subsequently dispersed to multiple storage locations and original data is recovered when needed sometimes with fewer numbers of dispersed data known as a *threshold*. There are many types of data sharing techniques, the foremost being the works of Adi Shamir [2] and George Blakeley [3] later to be known as a Secret Sharing Scheme. While the scheme provides the method of securing data without encryption keys in a dispersed storage, it has proved to be insufficient for use with big data as they are not very scalable according to Buchanan [39], hence the use in securing encryption keys mainly as it will be too processor intense to use in securing data [39]. Shamir and

Blakely’s works came to be known as a perfect secret sharing scheme. Other variants of data sharing of interest to us are Information Dispersal Algorithm by Michael Rabin [40], which tends to reduce the storage complexity experienced in Adi Shamir’s Perfect secret sharing scheme. Our interest is in its broader application for data dispersals since it has high-performance throughput when used to disperse data whether in-memory or in the cloud as it does not encourage redundancy with a trade-off on the perfect security of data. On a last note on secret sharing scheme is Social Secret sharing scheme (SSSS), which combines the features of Weighted Threshold Secret Sharing Scheme (WTSSS) [2], [41], [42], [43], [44] and that of Proactive Secret Sharing Scheme (PSSS) [45] in its design and concepts. We will be exploring its design principle of Sharing-Tuning-Recovery method in our future works while leveraging on this design to provide a self-organising system as proposed with a high level of data scalability in a multi-cloud architecture.

### 3.2.1 Shamir Secret Sharing Scheme (SSS)

Adi Shamir’s classic paper of “How to Share a Secret” is an example of a perfect secret sharing scheme. Thus implying that its information ratio is always unity for every process of secret sharing, as the size of the file is equivalent to each of the share created out of it. This implies providing perfect secrecy as shares less than the threshold cannot recover or learn of the secret, but a trade-off with performance. The larger the file size, the larger each share is and an increase in the number of participants in the sharing algorithm leads to higher storage overhead of the shares. With this, Shamir’s scheme is unsuitable for sharing large-scale data infrastructure according to [39], [31], [46], [5], [19], [13], [12], [18], [47] and [17].

SSS has been classified as a threshold scheme known as ($t$, $n$) scheme and it requires two conditions to be perfect: if, and only if, $t$−1 shares provide absolutely no information regarding the hidden secret. Also when the ratio of the length of the secret to the length of each of the shares (known as the information rate) is equivalent to 1, these views are shared collectively by [48] and [31]. Shamir’s PSS relies on the idea that on the principle that you can define a straight line with two points, three points for a quadratic equation, and so on, to give $t$ points to define a polynomial of degree $t$−1. Hence, a method for $t$-out-of-$n$ secret sharing can thus use a polynomial with a $t$−1 degree using a secret for the first coefficient, and then random values for the remaining coefficients. Next, find $n$ points on the curve and give one to each of the players. As a result, when at least $t$ out of the $n$ players reveal their points, there is sufficient information to fit a ($t$−1)th degree polynomial to them, in which the first coefficient is the secret.

The work of Shamir [2] and that of Blakely [3] which as ideal schemes are the foundation of Secret Sharing Schemes. Shamir's PSS is the foundation to a social secret sharing scheme as proposed by Nojoumian *et al.* [32] has applications and expansions in [34], [33] and [49]. This thesis tends to adapt its core principle of its 3-in-1 scheme of Share, Social Tuning and Recovery known as ***Sha, Tun and Rec*** with some modification in the Social Tuning protocol so as to be able to fit into our futurre research design in providing a redefined self-organisation different from their earlier proposal in [34] having found that their idea of social function has no place in disaster mitigation as some factors of disaster are not human rather than natural.

### 3.2.2 Information Dispersal Algorithm (IDA)

Rabin's [40] work is an example of non-perfect secret sharing scheme focused on dividing a secret $\boldsymbol{S}$ into $n$ pieces in such a way that anybody possessing shares less than the threshold $k$ can obtain the secret. Here, each secret $S_i$, $i \leq n$, is of size $|S|/k$, where $|S|$ is the size of the secret. The total sizes of all the secrets are [31]:

$$\left(\frac{n}{k}\right) \times |S| \qquad \text{Equation 12}$$

Thus, with the Rabin's Information Dispersal Algorithm, the storage complexity of a secret sharing system can be significantly reduced in comparison to Shamir's [2] perfect secret sharing scheme. But the security flaw in this method is that, if the data exhibits some pattern frequently, and that the attacker gets hold of $m < k$ slices, then there are great possibilities for him to get the secret $\boldsymbol{S}$.

Both the split and combine algorithms operate by performing matrix multiplication over the input. In the case of the split operation, the transform matrix has $n$ rows ($n$=*number of shares*) and $k$ columns ($k$=*quorum*), while in the case of a combine operation, the transform matrix has $k$ rows and $n$ columns. Either operation is described simply as the matrix multiplication [31]:

$$Transform\,Matrix \times Input\,Matrix = Output\,matrix \qquad \text{Equation 13}$$

The transform matrix must have the property that any subset of $n$ rows represents linearly independent basis vectors. If this is not the case then the transform cannot be reversed. However, understanding the requirement of linear independence is important in case a user-supplied matrix is provided and also for understanding the key parameter to the split/combine routines. A key is defined as a list of field elements (i.e., 8-bit, 16-bit or 32-bit values) [31]:

$$x_{1,}x_{2,} \dots, x_{n,}y_{1,}y_{2,} \dots, y_k \qquad \text{Equation 14}$$

whose values must all be distinct. If a key is supplied to the split routine, these values are used to create a Cauchy-form transform matrix [31]:

$$\begin{array}{cc} k\ column & \\ \begin{vmatrix} \frac{1}{x^1+y^1} & \frac{1}{x^1+y^2} & \cdots & \frac{1}{x^1+y^k} \\ \frac{1}{x^2+y^1} & \frac{1}{x^2+y^2} & \cdots & \frac{1}{x^2+y^k} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{1}{x^n+y^1} & \frac{1}{x^n+y^2} & \cdots & \frac{1}{x^n+y^k} \end{vmatrix} & n\ rows \end{array} \qquad \text{Equation 15}$$

Rabin’s IDA has shown so far as having high-performance throughput when used to disperse data whether in-memory or in the cloud as it does not encourage redundancy with a trade-off on the perfect security of data.

Lin and Chung [50] developed a variant of Information Dispersal Algorithm in order to correct their perceived security flaw in IDA in the design of a coding system. In order to correct this needed flaw, they designed a system called *An Efficient Information Dispersal Algorithm* using Fermat Number Transforms. In a bid to improve the computational performance, security and integrity of IDA, Lahkar & R [51] modified the scheme by combining the All-Or-Nothing Transform [52] with optimised Cauchy Reed-Solomon code, this they did by using a modified AONT as a pre-processing operation over the data.

An application of IDA in mobile networks aside, other application areas include in object storage in cloud [53] as provided in [54]. An evaluation of the effectiveness of this system called Reliable and Efficient Forwarding (REEF) using the Information Dispersal Algorithm (REEF-IDA) as against Reliable and Efficient Forwarding (REEF) [55] showed that the system REEF-IDA performed better than REEF by increasing the network throughput while decreasing both end-to-end delays and packet loss ratio.

### 3.2.3 Krawczyk’s Computational Secret Sharing

Hugo Krawczyk [11] proposed the Computational Secret Sharing (CSS) technique (a.k.a. *secret sharing made short*), which combines Rabin’s IDA [40] with Shamir’s PSS [2]. Data is first encrypted with a randomly generated key, using a symmetric encryption algorithm. Next, this data is split into $n$ fragments using Rabin's IDA with a threshold $t$ configured. In this case, the scheme is $t$ times more efficient than Shamir's PSS. The final step is to use Shamir's PSS to produce shares of the randomly generated symmetric key (which is typically of the order of 64 to 256 bits) and then give one share and one fragment to each shareholder.

A related approach, known as AONT-RS [52] as supported by [18], applies an All-Or-Nothing Transform (AONT) to the data as a pre-processing step to the IDA. AONT guarantees that any number of shares less than the threshold is insufficient to decrypt the data. It combines AONT with Reed Solomon (RS) in order to achieve high security in a reduced computational and storage overheads. AONT-RS is the backbone used in Cleversafe, which has since been bought over by IBM in October 2015.

### 3.2.4 Social Secret Sharing Scheme (SSSS)

Social Secret Sharing Scheme as proposed by Nojoumian *et al.* [33], [32], [34] involves three-fold constructions denoted by (***Sha, Tun, Rec***) implying Secret Sharing, Social Tuning and Secret Recovery. Its major difference to other threshold schemes is the social tuning, in which the weight of each participant is either increased or decreased with reference to participant's reputation.

There are three basic assumptions in SSSS, which states that to recover the secret, the total weight of authorised participants must be equal to or greater than the threshold, $\sum_{p_i \in \Delta} w_i \geq t,$ $\Delta$ is the set of participants and $w_i$ is the weights of the participants and $t$ the threshold; secondly, the weights of the unreliable participants must be less than the threshold, $\sum_{P_i \in \nabla} w_i < t,$ and finally, the weight of each participant is bounded by a parameter much less than the threshold, $w_i \leq m \ll t \; for \; 1 \leq i \leq n.$

#### 3.2.4.1 Share distribution

In SSSS each participant receives a constant number of shares; the adjustments are made in accordance to each participant's behaviours during each process and therefore it follows that the dealer generates a polynomial $f(x) \in \mathbb{Z}_q[x]$ of degree $t-1$, where $f(0) = k$, the secret, which is the constant term as stated earlier following [2] construction. Each participant $P_i$ thereafter receives a share; $1 \leq i \leq n$ with respect to his weight $w_i$ before the dealer leaves the scene [33], [32], [34].

$$\varphi_{ij} = f(\vartheta_{ij}) \text{ for } 1 \leq j \leq w_i \qquad \text{Equation 16}$$

Where $\vartheta_{ij} = im - m + j$ and $m$ is the maximum weight of any participant, which is zero for all at the initial stage.

#### *3.2.4.2* Social Tuning

Social tuning is the protocol used in SSSS to adjust the weights of participants as the process goes on as *ab initio* all participants receive an equal number of shares. The adjustments are made based on the participant's activities during each process of share creation and recreation measured in terms of active collaboration and response time during share request for reconstruction. Simply, put it involves three stages namely: adjustment, enrolment and disenrollment stages assuming that the weights are increased or decreased one by one. In adjustment stage, participant's availability and response time during the processes determine his reputation and thus result in the adjustment of their weight progressively or retrogressively. This is done using the following formulae [33], [32], [34]:

$$P_i(D): defection \Rightarrow w_i(p) = \left\lfloor w_i(p-1).(1-\tfrac{\tau}{2}) \right\rfloor \qquad \text{Equation 17}$$

Where $\tau = T_i(p-1) - T_i(p) \geq 0$ is the coefficient of weight reduction for non-collaborative participants and whenever the weight of any participant is $w_i(p) = 0$, the participant $P_i$ is removed from the scheme. In the same vein, enrolment stage is activated whenever the weight of any active participant is to be increased by any value, at this point all the participants collaborate to generate a new share on the initial secret sharing polynomial for the participant, [33] has an elaborate details on enrolment protocol. Lastly, at the disenrollment stage, inactive participants are dropped when all the active participants gather to update their shares without the participation of the inactive ones. When this is done all shares are updated to be on a new secret sharing polynomial $f(x)$ but because the inactive ones do not participate, their shares remain on the old secret sharing polynomial $f(x)$ making them invalid and therefore can no longer participate in the secret reconstruction. The active or inactive status or reputation of each participant is known by using the trust calculation method proposed by [49] to calculate the average of the trust values in order to compute a participant's reputation after each secret sharing process [33], [32], [34].

$$T_i(p) = \frac{1}{n-1}\sum_{j\neq 1} T_i^j\ (p) \qquad \text{Equation 18}$$

where $-1 \leq T_i(p) \leq +1\ and\ T_i(0) = 0$. A new Trust Function has also been proposed for Social Secret Sharing in Cloud Computing by [34].

#### 3.2.4.3 Secret Recovery

Just as in Shamir [2] authorised participants following earlier stated rules are able to recover the secret using Lagrangian interpolation once the condition $\sum_{p_i \in \Delta} w_i \geq t$ as stated earlier is met. The participants $P_i \in \Delta$ contribute their shares $\vartheta_{ij} for\ 1 \leq j \leq w_i$ to recover the secret $f(0) = k$.

Eslami *et al.* [56] in their work, proposed a variant of social secret sharing scheme that proved to be more efficient in terms of share size, communication and computational complexities by using Birkhoff interpolation scheme as opposed to Lagrangian Interpolation used in Social Secret sharing scheme but admits that social secret sharing scheme is better in terms of social tuning and recovering of secret compared to theirs. While Traverso *et al.* in [57] provided a framework called Adaptive Social Secret sharing (AS3). The framework is a variant of social secret sharing that uses a dynamic approach in secret sharing and recovering. One major difference is in the dynamism of allocation of shares thus provides an efficient and optimal storage system than Social secret sharing scheme. The scheme also demonstrated through a proof of concept a different way of computing trust value as well as initialising such for newcomers.

Social secret sharing scheme as proposed has its social tuning function based on the calculation of trust function as well as used perfect secret sharing in breaking data into shares and recovering it. Using such scheme for large-scale data infrastructure is unsuitable and determining cloud 'behaviour' based on results of two capacities such as response time and rate of availability will be relying on incomplete information in making a critical decision such as adjustment, enrolment and disenrolment of a participant. This is because cloud 'behaviours' can be affected by its multi-tenancy nature and hence the reason throughput fluctuates. So, it will be better to rely on more than two capacities such as throughput, reliability, transaction speed and Integrity. Moreso, using current statistics without looking at future ones may not be a correct approach as *behaviours* that are subjective can change.

### 3.2.5 Secret Sharing and Multi-Cloud Architecture

National Institute of Standards and Technology (NIST) define Cloud computing as a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction [58], others in contrast define cloud as an elastic execution environment of resources involving multiple stakeholders and providing a metered service and multiple granularities for specified level of quality [59], [60], and [61].

The major concerns of moving towards cloud-based storage have been security and availability of data when accessed. Several researchers defined that a multi-cloud storage platform might improve security and resilience [62], [63], and [64]. Based on these and to ensure effective data splitting and security of stored data, a secret sharing algorithm such as Shamir secret sharing has been proposed as an efficient scheme for multi-cloud storage [4], [65], [65], [66], [67], [68], [69] and [70]. Several other methods use Backup and Restore while some other ones replicate data at different storage areas using encryption to safeguard them. This type of methodology can be costly (in terms of network bandwidth and storage facilities), and not safe in the case of poor key management.

### 3.2.6 Security Limitations of Secret Sharing Schemes

Recent developments in information and communication technologies infrastructure stipulate a rapid growth of electronic data exchange. All the same, it is common for the public and private institutions as well as the industries to outsource massive electronic databases to storage centres. The cloud computing technology allows the users to work with such centres without even knowing their internal structure. However, storing all the data in one centre creates a single point of failure and raises privacy and availability concerns, especially in the sense of disaster preparedness and recovery. Secret sharing is a cryptographic technology, which allows us to address both privacy and availability issues simultaneously [71].

However, in Dautrich & Ravishankar’s work [72], titled: *Security Limitations of Using Secret Sharing for Data Outsourcing* demonstrated that the claims made by three different works, [73], [74], [75] that when $k$ shares that are required to recover a secret or more collude, adequate security of the scheme remains intact as long as a prime $p$ and a vector $\mathbf{X}$ used by the secret sharing algorithm are kept private by describing and implementing an attack that reconstructs all secret data when only $k+2$ secrets are known initially. With this experiment, they were able to recover a hidden 256-bit prime for $k \leq 13$ servers, or an 8192-bit prime for $k \leq 8$ in less than 500 seconds.

Moreso, Tompa & Woll [76] in their work: *How to Share a Secret with Cheaters* identified a flaw in Shamir threshold scheme that makes it susceptible to attack by Cheaters by assessing the impact of an active adversary who takes the form of a participant but maliciously submits a false share during a reconstruction phase. Take for example; some participant $P_i$ submits a false share $\lambda_i$ instead of a correct share $f(x_i)$. This without mincing words will result to the prevention of an honest participant from learning the correct secret thus failing to alert the other participants that they have not reconstructed the correct secret and this, in turn, allows the adversary to learn the correct secret (by exploiting knowledge of $f(x_i)-\lambda_i)$ [38].

## 3.3 Secret Sharing/Data Striping: Applications

In storage technology, data striping is used to break data into slices and subsequently dispersed to multiple storage disks. Striping technique as applied to several disks comes to bear when a data processing device requests data more frequently than a single storage device can handle. Its application is spread across the accessing of multiple concurrent devices as it increases total data throughput. One of its usefulness is in balancing Input/Output load across an array of disks such as in Redundant Array of Independent Disks (RAID).

Beyond this, secret sharing as a method of creating shares of data, disperse to multiple storage devices and recover when needed has been found better in security and availability of data. It has two major areas of application – data sharing and key management as has been applied successfully in the collaborative and secure sharing of healthcare data in multi-clouds in [12] where secret sharing scheme is used alongside attribute-based encryption in providing selective access authorisation in order to disperse data across multiple clouds. In [13], Tatiana and Benjamin demonstrated how they combined Shamir secret sharing scheme [2] and Rabin's Information Dispersal Algorithm (IDA) [40] for use in distributing health data to multiple cloud providers as fragments, which also have the ability to provide higher redundancy, security and privacy needed for safekeeping of the data in cloud system, while Shor *et al.* [4] used same in managing encryption key successfully.

Loruenser *et al.* [25] presented an architecture for secure cloud-based data sharing known as ARCHISTAR based on secret sharing scheme. The focus of the system is on providing adequate confidentiality to data; make it available against any active attacks as well as robust even in the face of failures. There are other research solutions based on different variants of secret sharing schemes and multi-cloud architecture that give credence to its resilience in the face of failures, data security in a keyless manner, such as:

Ukwandu *et al.* [46] worked on a research article titled - *RESCUE: Resilient Secret Sharing Cloud-based Architecture,* which presents an architecture of a system that is capable of implementing: a keyless encryption method with in-built failover protection. It aims to overcome many of the current problems within Cloud-based infrastructures, such as the loss of private keys, and inherent failover protection. While RESCUE provides an architecture for a resilient cloud-based storage with keyless data security capabilities using a secret sharing scheme for data splitting, storage and recovery; Alsolani & Boult in CloudStash [17] also relied on the above strengths to prove security of data using secret sharing schemes in a multi-cloud environment; Fabian *et al.* [12] provided empirical evidence of resilience and robustness in data sharing using secret sharing scheme in a multi-cloud environment and Buchanan *et al*. [31] predicted that future internet will be a world of secret shares.

In all these, secret sharing was seen as limited in scope and therefore is unfit for use in sharing large data infrastructure, supporting the initial evidence provided by Buchanan [39] on limitations of secret sharing scheme applications.

Lastly, of major interest to us is the use of data striping by International Business Machine (IBM) for Cloud Object Storage [53]. The system uses data striping as an innovative approach for storing a large volume of data while ensuring security, availability and reliability. It uses a modified Rabin's IDA by introducing symmetric Reed Solomon to stripe data and distribute same across multiple data centres securely. No single node contains a complete copy of the data and only a subset of nodes needs to be available in order to fully retrieve the data on the network. This work previously known as Cleversafe before being bought over by IBM in October 2015 provides better alternative to the idea of data replication and remain a pioneering work in our area of interest in combining data striping with multi-cloud architecture in achieving reliance, availability, security, resilience and scalability for big data but failed to interpret robustness in the area of self-organisation, rather use canary for corruption checks and encode data with encryption key while dispersing and this according to Schneier [30] is more tedious than encryption itself.

## 3.4 Performance Evaluation of Secret Sharing Schemes

Secret sharing scheme has two main application areas: data sharing and recovery and key management. While proponents of use for data sharing are of the view that using such is scalable; secure, resilient, improves data confidentiality, integrity and availability. In the other hands, the other proponents argue that secret sharing scheme has inherent bottlenecks that make it unsuitable for use in sharing and recovering large-scale data infrastructure as it was originally proposed for use in managing encryption keys. The performance evaluation of these positions will be reviewed here to ascertain the necessary bottlenecks, knowledge gaps and suitability of both positions for use in sharing large-scale data infrastructure in multi-cloud architecture.

### 3.4.1 Scalability

This thesis defines scalability of data sharing methods as the ability of a method to continue production even when file size increases exponentially, this will form the basis of our review on the concept of scalability of secret sharing scheme in data sharing and recovery. The essence is to ascertain the available knowledge gaps in this area of application of secret sharing scheme.

Alsolami and Boult in [17] built a system called CloudStash, which posits that secret sharing is more suitable in securing data not encryption keys and by performing some experiments, showed that CloudStash is faster when used in small sized file. But with large file sizes such as 10MB, admitted that the overheads grew exponentially and suggests that the perfect secrecy of using secret sharing should be a good trade-off in this regards. In all CloudStash did not show enough evidence of scalability and fit for use in large-scale data infrastructure, as results proved otherwise even when performed in a multi-threaded system.

Ermakova and Fabian [13] and Fabian *et al.* [12] in their work used combination of secret sharing scheme with other cryptographic primitives in securing healthcare data through shares creation, dispersal and recovery in multi-clouds. In all, Abdallah and Salleh [19] and [27] showed that secret sharing has inherent bottlenecks that make it not suitable for use in big data as increase in the number of participants increases data sharing overheads, while increasing the threshold increase data recovery overheads and hence not scalable.

### 3.4.2 Resilience

The works of Abdallah and Salleh [19] and [27] provided information supporting the evidence that when the number of shares available for recovery is less than the defined threshold, implying that $n < m$, for $n$ as the number of participants and $m$ as the defined threshold, secret recovery is impossible and hence a proof that the use of secret sharing has limited level of resilience. In view of this, a review of the concept of resilience in secret sharing needs to be considered as an ability to continue production at different rates of cloud outages within a reasonable extent.

### 3.4.3 Key management

Works of Shor *et al.* [4], and [18], [17], [25], [78] used secret sharing scheme in their key management and attest that using secret sharing scheme in key management is robust, resilient, and secure. It has shown as the ideal method for use in multi-cloud environments but did not extend their research on the impacts of cloud outages on key management. This has been notified as a knowledge gap they failed to address.

## 3.5 Sharing a big data

Shor *et al.* [4] believe that the only way to secure sensitive data before storing it in the cloud is to encrypt before storage. With an evaluation of the inherent trade-offs of securing data in remote storage as well as an end-to-end analysis of the current methods of securing data using secret sharing scheme, encryption-based schemes on a local cluster storage device and multi-cloud environments. Their results suggest that the bottlenecks in securing data has moved from that of computational overheads of encoding and random data generation to network, storage and availability as a result of hardware accelerated encryption methods and hence concluded that data encryption and management of keys with an efficient secret sharing scheme is optimal for multi-cloud environments. These claims run contrary to the works of Resch and Plank, known as AONT-RS [52] as supported by [18], it applies an All-Or-Nothing Transform (AONT) to the data as a pre-processing step to the IDA. AONT guarantees that any number of shares less than the threshold is insufficient to decrypt the data. It combines AONT with Reed Solomon (RS) in order to achieve high security in a reduced computational and storage overheads.

While Ermakova and Fabian [13] and Fabian *et al.* [12] used combination of secret sharing scheme with other cryptographic primitives in securing healthcare data through shares creation, dispersal and recovery in multi-clouds,  Alsolami and Boult in [17] built a system called CloudStash, which posits that secret sharing is more suitable in securing data not keys. Through experimental results showed that CloudStash is faster when used in small file significantly but in large file sizes such as 10MB, admitted that the overheads grew exponentially but suggests that the perfect secrecy of using secret sharing should be a good trade-off in this regards. Shor *et al.* [4] concerned themselves with the protection of big data and of encryption key using multi-cloud environments forgetting that using a single point storage for data is susceptible to single point of failure. It is only an available data that can be decrypted, therefore using data fragmentation seems more promising in safeguarding sensitive data as well as sharing big secret than their approach in cloud-based system.

## 3.6 Cloud-based data storage

Cloud data storage system is not new in computing as it is one of the widely deployed Infrastructure as a Service (IaaS) in cloud computing. The evolving technology has been the use of multi-cloud rather than single cloud for cloud storage in order to improve on data availability, confidentiality and integrity [9], [12], [13], [25], [17] and [46]. A lot of characteristics of the cloud storage make it attractive from being scalable for a big data infrastructure, to cheaper service of the pay-as-you-use model, availability, flexibility [79], and disaster management [80], [81], [82], [9], and [25]. But there are some concerns in areas of data security, privacy and confidentiality due to multi-tenancy storage approach used in the cloud. Khoshkholghi *et al.* [79] classified cloud data storage into four layers: physical storage, infrastructure management, application interface and access layer. Some practical solution proffered to some issues with cloud storages raised earlier show that progress have been made to ensure data protection as well as redundancy in cloud storage.

As threats to data integrity consist of malicious third party occurrences and hosting infrastructure weaknesses, issues like Proof-of-Readability (POR) and Proof-of-Data Protection (PDP) protocols are well studied in the literature by [83], [84]. There is also a compact improved version by [85] and for high-availability (HAIL) by [85]. Wang, *et al.* [86] and [87], [88] worked on auditing the security of cloud storage, which suggests that an interface layer can help user assess risks. Tang, *et al.*, [89], and [90] worried that users' data when deleted can still be restored through a backup version by

CSPs and therefore presents a work called FADE, which is a Secure Overlay with File; Access Control and Assured Deletion.

To ensure data availability, [91] applauded the combination of *strong consistency, global partitioned namespace and disaster recovery approach* of Windows Azure Storage in ensuring availability of multi-tenancy environment. Figure 6 shows how data stored in the cloud are accessed commonly through the web service APIs and that remains a point of dissimilarities between traditional and cloud storages, while Figure 7 shows the cloud storage reference model as designed in [92].

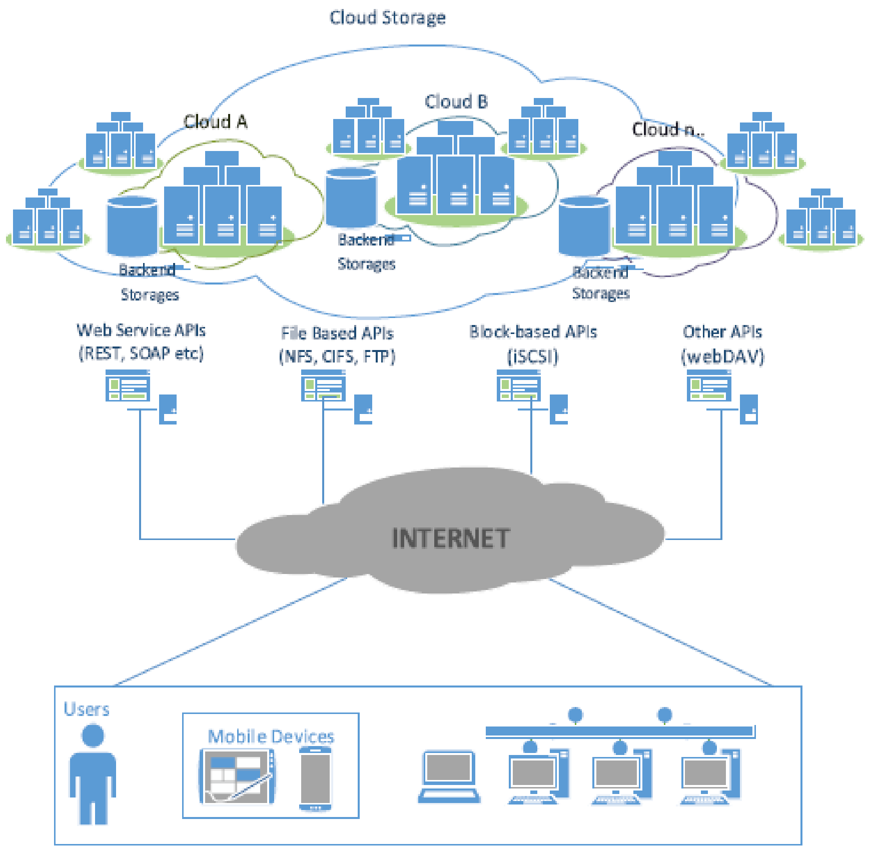
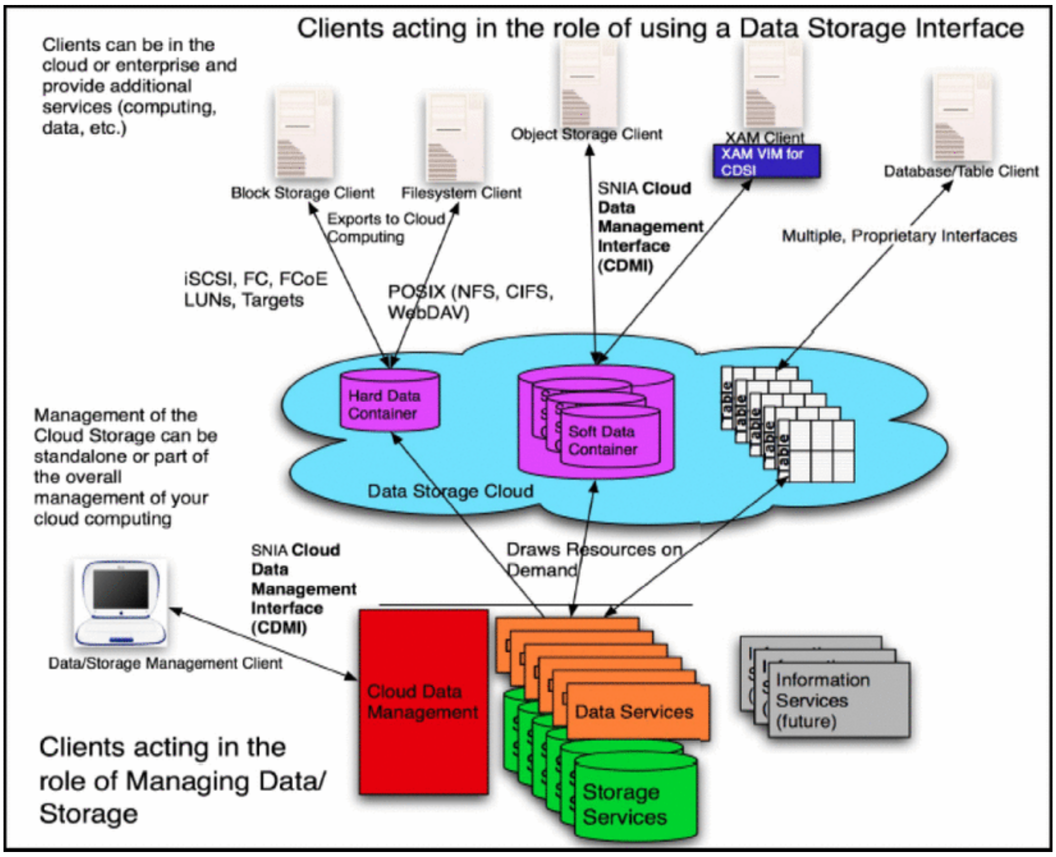


Figure 6: Cloud Storage Access Method [93]



Clients acting in the role of using a Data Storage Interface
Clients can be in the cloud or enterprise and provide additional services (computing, data, etc.)
Block Storage Client
Filesystem Client
Object Storage Client
XAM Client
XAM VIM for CDSI
Database/Table Client
Exports to Cloud Computing
iSCSI, FC, FCoE LUNs, Targets
POSIX (NFS, CIFS, WebDAV)
SNIA Cloud Data Management Interface (CDMI)
Multiple, Proprietary Interfaces
Hard Data Container
Soft Data Container
Table
Data Storage Cloud
Management of the Cloud Storage can be standalone or part of the overall management of your cloud computing
Draws Resources on Demand
SNIA Cloud Data Management Interface (CDMI)
Data/Storage Management Client
Cloud Data Management
Data Services
Information Services (future)
Storage Services
Clients acting in the role of Managing Data/ Storage

Figure 7: Cloud Storage Reference Model [92]

### 3.6.1 Cloud-based Key Management System

The security of stored data in cloud is crucial in cloud computing as Wang *et al.* [94] posit that such necessitates the need for the design of a key management scheme that is reliable for safe computing in the cloud. Rao [95] agrees that key management is not standardised optimally in the cloud. Rao and Selvamani [96] are of the view that having only authorised users to have access to the decryption key is the best management. While Zissis and Lekkas [97] suggest that having a trusted third party is the way to go. In order to achieve this objective as opined, two major methods suffice – keyless and In-house-key-storage management system. In keyless system [4], [18], [17], [25] and [78] used secret sharing scheme, but Resch and Plank [52] dispersed encryption key with the encrypted data in cloudlets, while [87], [98] proposed key aggregate key management system in managing In-house-key-storage. In all, it shows how important key management is in cloud-based data storage. From the methods presented, the keyless type provides a system that prevents key loss, theft and leakages and as such is resilient, reliable and as well provide confidentiality and availability needed in a key management system for a robust cloud-based storage.

### 3.6.2 Cloud-based Disaster Recovery (DR) System

The cloud disaster recovery system is entirely different in approach to traditional disaster recovery system. A cloud-based system takes an integrated approach in which the virtual server gets bundled with the operating system, applications, software patches and data. Furthermore, the backing up and copying of the entire server to an off-site data centres through virtualisation take as little as minutes [6].

The virtual server does not depend on a particular hardware, which makes it easier to transfer safely and accurately from one data centre to another, the operating system, applications, patches and data without reloading each component of the server. The advantages of Cloud-based DR over the traditional type range from reduced recovery time and complete data accuracy during data restoration and recovery. These are made possible because of its ability to implement full network replication, the synchronisation or mirroring of Virtual Machines (VM) at a remote site to ensure failover in the event of failure of the original site.

Cloud-based DR as in all cloud-based systems provides low-cost DR solutions because of the “pay-as-you-use” model. Cloud-based DR provides its subscribers both shared and dedicated DR services. Based on customers’ choice, the benefits can accrue but in all, cloud-based DR offers low-cost services compared to traditional systems. It is this “resource-on-demand” and a high degree of automation that made cloud-based DR very attractive. Despite attractive economics, Wood *et al.* [99] argue that increased latency is a major barrier in using cloud data centres for DR, as other servers could have a large geographical separation from the primary site, which in no small measure could affect communication between them adversely. The limitation of data owners from having control over their data placement worsens this scenario, they argued.

In the same vein, Ji *et al.* [100] and [79] opined that the use of synchronous replication by Cloud-based DR does not in any way help every data write as the wide-area latency has negative impacts on its performance, forcing system administrators to consider asynchronous replication. This often trades-off loss of data for performance by replicating a consistent “snapshot” to the backup site. Asynchronous replication positively impacts performance as the primary site can come up even before the replication completes. However, this can lead to loss of disk writes at the primary site subsequent to the last replicated snapshot in case of a disaster.

### 3.6.3 Benefits, Challenges and Solutions

As DR is aimed at business continuity, its benefits are: assets and inventory management, network management, task redundancy, cost saving and ability to test your plan in various scenarios ahead of time. But with the introduction of cloud-based DR, the dimension of its benefits could take a different turn as cloud systems deliveries are much faster and can provide multi-site availability at a fraction of the cost of traditional disaster recovery.

Figure 8 provides insight into the graphical explanation of the benefits of cloud DR systems. The red arrow represents cloud-based DR systems, it shows the cost-effectiveness of cloud DR with significantly faster recovery times [8]. Cloud resources have made disaster recovery cost-effective and attractive as it provides quick data recovery by providing rapid failover and failback capabilities to the primary site in the face of a disaster [8].

Despite these benefits, there are some common challenges accustomed with cloud-based DR mechanisms, ranging from dependency according to [79] and [101]. This arises as customers depend on cloud service providers for DR services for lack of direct control over their data thereby creating a serious challenge of selecting a trusted service provider. Cost, though cheaper in cloud-based DR but appears in three phases as initial cost, ongoing cost and cost of the potential disaster. Others are failure detection, data security in the face of disaster, replication latency; centralised data storage and lack of redundancy during failover as the primary site remains down until brought back during failback phase.

But these are not without some proposed fine-grained solutions based on Redundancy and Backup strategies as proposed by [101] on using Local Backup; [102] and [103] on the application of Geographical Redundancy and Backup; [104], proposed the use of Inter-Private Cloud Storage; [105], opted for Resource Management (use of enhanced technology) for data recovery in storage clouds, while [106] talked of Secure-Distributed Data Backup as measure to protect data in the event of disaster. Pipelined Replication has been proposed by [107] as a way to team up the performance of asynchronous replication with the consistency of synchronous replication in the face of disaster. Nakajima *et al.* [108] discussed the use of Scale Up/Down technique while Aghdaie *et al* [109], proposed Dual-Role Operation.

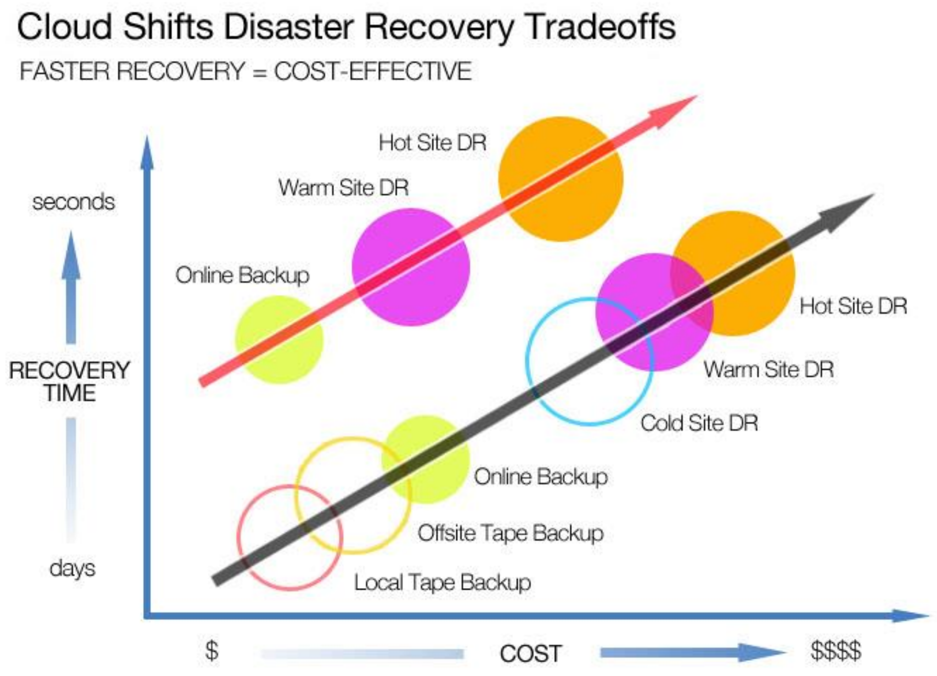


Figure 8: Disaster Recovery Trade-Offs in Cloud DR [8]

### 3.6.4 Fine-Grained Cloud-based DR solutions

The evaluation of fragmented secret share system will also look at its potential application area – cloud-based disaster management and hence a review of some fine-grained current solutions in this area. The essence is to elicit knowledge on their strengths and weaknesses that provide the need for FSSS.

#### *3.6.4.1* DR-Cloud

Gu *et al.* [9] relied on data backup and restore technology to build a system proposed to provide high data reliability, low backup cost and short recovery time by utilising co-operative resources of various cloud service providers with various parameters using multiple optimisation scheduling as strategies in balancing the objectives of disaster recovery. The system is built of multi-cloud architecture using Cumulus [110] as cloud storage resources. Thus, providing the need for further studies on the elimination of system downtime during a disaster, as well as provide consistent data availability as there is no provision for such in this work, rather concentrated on data reliability, low backup cost and short recovery time.

#### 3.6.4.2 Cloud Standby

The use of fully operational standby sites and having the systems periodically updated is an approach Lenk *et al.* [111] opined here as a way of being at alert against disasters. Thus, by describing the architecture for a novel approach in establishing standby sites, known as warm sites in disaster recovery that replicates a distributed system in the cloud to another cloud, they argued is a missing link in current literature. Hence in their architecture - they argued that it provides a better warm standby approach for setting up and updating a standby system in the cloud. In order to reduce the recovery time, they developed a deployment method that allows providers an independent and automated distribution system in [112] known as A Model-Driven Deployment Method for Disaster Recovery in the cloud.

No doubt, their system is intended to provide a backup system in case of disaster. Even with their deployment method, the certainties of downtime cannot be overemphasised and hence flawed in providing a resilient system that can mitigate disaster rather than recovery after the hazards must have taken place.

#### 3.6.4.3 Adaptive Remus

Cully *et al.* [67] described a system that provides software resilience in the face of hardware failure (VMs for Virtual Machines) in such a manner that an active system at such a time can continue execution on an alternative physical host while preserving the host configurations by using speculative execution in replicating either processor-intensive applications or communication-intensive applications at a fixed time interval. The strength lies in the preservation of system's software independently during hardware failure but lags in the area of performing Replication by adding fault tolerance into the VM at fixed intervals. This creates confusion between processing-intensive applications and network-intensive applications maximisation as longer intervals benefits the former, while shorter intervals the latter thus creating a dynamically adaptive time interval for optimal utilisation of both resources becomes a knowledge gap Silva *et al.* [113] proposed to achieved by designing Adaptive Remus, which dynamically adjusts the replication frequency according to the characteristics of running applications.

It is obvious with Adaptive Remus that speculative execution is still involved as time delays and other factors can lead to miscalculation of resource replication time and hence downtime is inevitable but may be minimised.

#### 3.6.4.4 MCES

Dong *et al.* [114] described an architecture known as Multi-cloud-based Evacuation Services for Emergency Management (MCES) that is based on instantiating multiple instances at different states as a way of mitigating the cloud-based disaster. Their work is similar to that of Chu and Wu [115], Chu and Wu [116] and Chen *et al.* [117]. Chu and Wu provided a hybrid system that combines cloud resources with mobile phones. The cloud serves for routine task computing while sensor information that provide best evacuation routes for data are collected using mobile phones and these are done irrespective of an emergency situation or not. Chen *et al.* concentrated on mobile cloud computing using smart-phones. Using their proposed system mobile phones are used to collect sensor information for which evacuation route are found based on sensor information and user location information.

Dong's have instances status cycle scheduling framework organised in three layers respectively with their different status: service, sleeping, and snapshot. The multi-cloud-based evacuation services (MCES) architecture maintains basic monitoring and maintenance services during times of normal activity but quickly scales up service capacity during an emergency; these were achieved using different sets of algorithms. It is obvious that MCES interest lies solely on maintaining system infrastructure by instantiating a new VM when one goes off and of course, the instantiation takes time depending on the software systems (OS and Applications) to be reinstalled. Thus, proving a point that MCES is designed for disaster recovery with its attendant downtimes depending on the time it takes to re-instantiate a new VM after the cloud disaster.

## 3.7 Weaknesses and research challenge of present methods

Several variants of data sharing techniques that incorporates secret sharing scheme designed for use in multi-cloud environment lay claim to scalability and resilience. While two research directions exist in the use of secret sharing scheme in data sharing, one is of the opinion that its implementation is primarily for protection of encryption key, while the other believes it is capable of being used to protect data in a keyless manner.

According to Abdallah and Salleh [19] and [27] secret sharing has inherent bottlenecks that make it not suitable for use in big data as increase in the number of participants increases data sharing overheads, while increasing the threshold increase data recovery overheads and hence not scalable. Also, their works in [19] and [27] provided information supporting the evidence that when the number of shares available for recovery is less than the defined threshold, implying that $n < m$, for $n$ as the number of participants and *m* as the defined threshold, secret recovery is impossible and hence a proof that the use of secret sharing resilience is limited in scope.

In all, there is a gap in knowledge on what constitutes scalability and resilience of these methods of data sharing in multi-cloud environment. Establishing a standard through which these capacities can be evaluated remains a research challenge needed to be addressed.

## 3.8 Conclusions

This chapter reviewed current literature around areas FSSS is premised. By reviewing literatures in Data Sharing and Recovery schemes such as that of Shamir [2], Rabin IDA [40], Krawczyk [11], AONT-RS [52] and Social Secret sharing scheme [32]–[34], [49] knowledge on their potentials, weaknesses and applications in data security with reference to cloud data storage security has been provided. The essence is to bring attention and focus on why these limitations made them more suitable in key management than data sharing and dispersal in a multi-cloud architecture especially Shamir's Perfect Secret Sharing scheme. The knowledge gaps found in the work of Shor *et al.* [4] and the performance evaluations of secret sharing scheme with respect their scalabilities, resilience and key management together laid the foundation on which FSSS will be built. Reviews on Cloud-based Data Storage system have provided knowledge on the evolving technologies in the use of this cloud resources from single cloud to multi-clouds storage to improve on data availability, confidentiality and integrity [9], [12], [13], [25], [17] and [46] and some concerns bothering on key management, privacy, security and confidentiality due to multi-tenancy storage approach used in the cloud.

With further exploration on works that ensure data availability autonomously or otherwise regardless of hardware failures, corrupted physical disks or downtime as well as their several key management methods this thesis has been able to provide insight why cloud outages persist, the most attractive key management method in cloud-based storage and establish the need for a robust system that redefines disaster management. The literature has shown that secret sharing is unsuitable in dispersing large-scale data infrastructure rather better used in key management with reference to Shamir's scheme. The use of multi-cloud in conjunction with secret sharing scheme have proved to be resilient, reliable, improves confidentiality and security. It is therefore necessary to design a method that combines high data scalability in multi-clouds with secret sharing scheme for key management with the aim of redefining robustness in cloud-based disaster management.

The next chapter presents the design of FSSS, whose method aims at providing a scalable way of sharing any amount of data within cloud-based architecture and hopes to redefine robustness and resilience using the concept of self-organisation in future works. It combines data fragmentation and secret share system known as a fragmented secret share system.

# 4 Experimental Design

## 4.1 Introduction

This thesis is focused on the evaluation of fragmented secret share system against similar methods in relation to scalability, key management and resilience. This chapter is therefore aimed at bringing to fore the overall system architecture, design principles, and methodology. The methodology comprises of the experimental setup and evaluation frameworks. The nature of FSSS design is on software and hardware systems and fully cloud-based. This thesis therefore uses this chapter to lay out the detailed overall architectural design, core design principles and steps used in the experimental setups, the aims of the experiments, justification and metrics used in each experiment. RESCUE developed two major evaluation frameworks in equations 19 and 20 that will guide the overall evaluations based on scalability and resilience with reference to similar methods. These two, especially resilience form the background on the evaluation of the key management, while both are the core that informs the potential application areas of mitigating cloud-based disaster.

## 4.2 Design Architecture

Figure 9 shows a high-level architectural design of FSSS. It is built as a method that accepts user's input and uses same to determine appropriate fragment size for which the file is broken into fragments. With the user's choice of share policy, the number of cloud providers (cloudlets) that will participate in the operation is determined. The fragments automatically created are each encrypted and shares created out of it based on share policy. The storage of these shares, as well as the encrypted fragments (chunks), is done in the cloudlets in such a manner that when the file is required, the key shares are recovered and each recovered key used to decrypt corresponding encrypted chunk and finally recombined the original file, after which file checksum is performed before despatching it to the file owner. Major components are input and output terminals, FSSS engine is made up of file-splitter, key generator, share and fragments creation, storage, recovery and file-combiner alongside checksum and metadata, which serves as the database for storing share, fragments and file information.

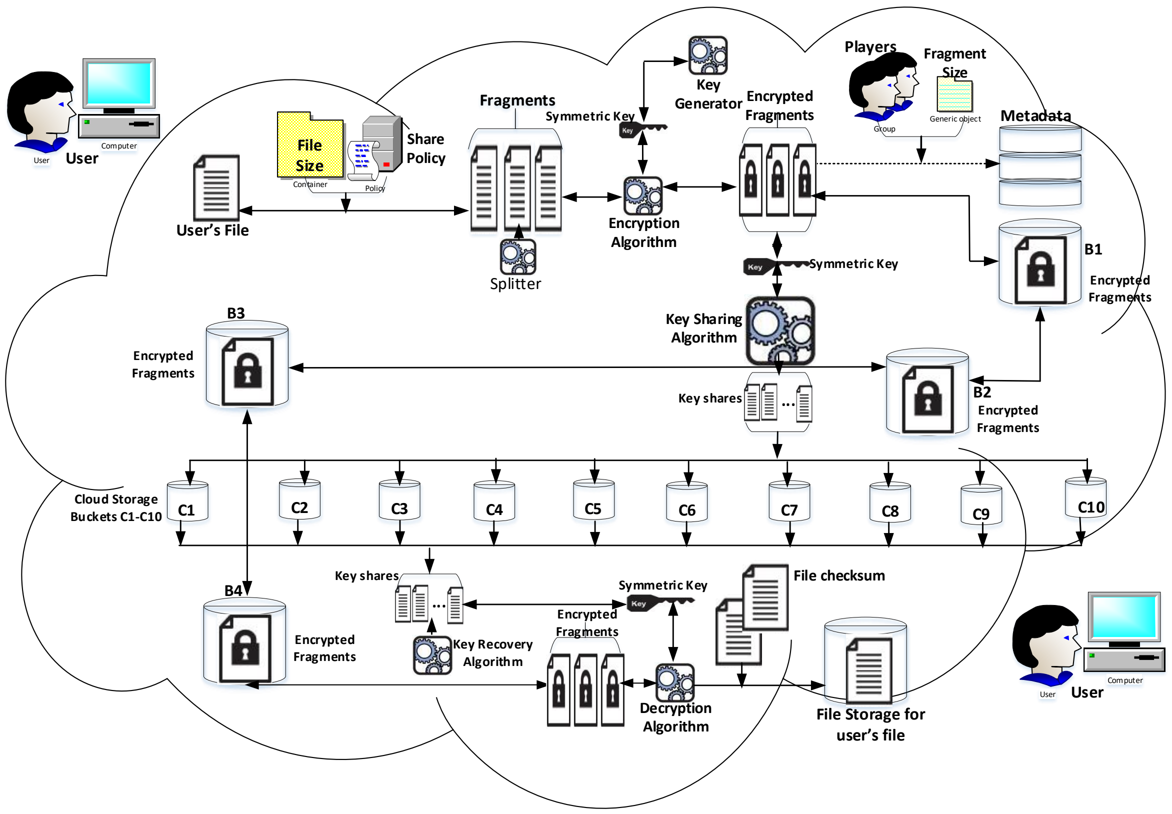


Figure 9: Overall FSSS Design Architecture

## 4.3 Design Principles

FSSS is designed to meet these objectives: First, to provide a method that is fit for sharing large data infrastructure and yet resilient, robust, readily available and reliable. Previous literature in [19], [5], [39] and [17] showed the strength and limitations of secret sharing scheme in data sharing and recovery. Thus, showing the need to incorporate data fragmentation with secret sharing scheme in order to provide high level scalability for file of all sizes and types. Secondly, provide information on best practices that will ensure quick, efficient, secure and scalable operations through combination of fragments, file sizes, share policies, and cloudlets. Finally, use the above qualities in redefining cloud-based disaster management away from current practices of recovery after cloud outages as seen in [25], [113], [112], [107], [81] to disaster mitigation to forestall losses.

### 4.3.1 System components

The system is made up of four main components:

1. **User Management:** Login details, file size options, share policy options, metadata creation.

2. **File Fragmentation: f**ragments creation, key generation, encryption, share creation, share dispersion, storage and then followed by numbers 4 and 5 below.
3. **File Recreation:** Login details, metadata retrievals, encrypted fragments recovery, key share recovery, key recreation, fragments decryption, file recreation, checksum and storage, followed by numbers 4 and 5 below.
4. **Cloud Behavioural Computations**.
5. **Agent** – Analyses and future behavioural predictions, clean-ups, self-organisation if needed (future works).

## 4.4 System Design Methodology

FSSS methodology entails steps and procedures taken in conducting experimental tests, metrics used, and the evaluations in order to ascertain the significance, and weaknesses with respect to other similar methods in areas such as scalability, resilience, which affects key management system. Four main experimental tests will be outlined, the metrics, alongside procedures taken and the relevant evaluations done.

### 4.4.1 Experimental setup

#### Scalability

**Experiments A**: Break file of sizes 10KB, 100KB, 1MB, 10MB, 100MB, 1GB into fragments.

Steps taken with graphical details in Figure 10:

(a) Use 10KB as defined block fragment size to break the above file sizes each. Repeat same with 100KB, 1MB, 10MB, 100MB and 1GB sizes.

(b) Encrypt each fragment with AES 256-bit key length as generated by the key generator for each fragment.

(c) Using *n* from chosen share policy generate the equivalent number of cloud service providers (CSPs) and initialise for storage of key shares as shown in Figure 11.

(d) Initialise three CSPs for encrypted fragments storage.

(e) Disperse encrypted fragments to 3 default CSPs initialised for it.

(f) Disperse key shares to the specified CSPs in accordance to *n*.

(g) Return all IDs (fragments and key shares to metadata table) as in Table 1.

(h) Generate user's ID and file details.

(i) Input user's ID, use same to retrieve the encrypted fragments, decrypt and recover the original file as shown in Figure 12.
(j) Calculate cloud behaviours using metrics in Tables 2 and 3.

**Aim:** To measure the time taken to break each file into block fragments, disperse and recover the original file in relation to share policy.

**Metrics:** file size over time taken to process in relation to share policy.

**Justifications:** This thesis chose minimum and maximum file sizes, alongside ranges of share policies to be able to have a controlled experiment within a specific time frame. The incremental nature is to help find appropriate results that fit into the metrics being evaluated. The block sized fragments are chosen to enable this thesis evaluates the scalability of block and chosen optimum fragment sizes at different share policies. All files are byte streams to provide a neutral ground for all file types. Encryption method is AES 256-bit key length with Electronic Code Book (ECB), the basic method as byte streams broken into fragments were being encrypted and hence did not consider using Salt and Initialisation Vector (IV).

**Tests conducted**

**Test One: Tests of Share Policies, File Sizes and Fragments**

**Test 1a:** Fixed share policies, varied file sizes, varied number of fragments and system overhead

**Test 1b:** Varied thresholds, varied file sizes, same fragment size and system overhead

**Test Two: Tests File Sizes, Fragments, and Share Policies**

**Test 2a:** Varied file sizes, fixed share policy, varied number of fragments and system overhead**.**

**Test 2b:** Varied file size, varied share policy, varied fragment sizes and system overhead.

**Experiments B**: Break file of sizes 10KB, 100KB, 1MB, 10MB, 100MB, 1GB into fragments.

Steps taken with graphical details in Figure 10:

(a) Using file size of 10KB, calculate 15% of file size and use as fragment size and break the above file sizes each. Repeat same with 100KB, 1MB, 10MB, 100MB and 1GB sizes.
(b) Encrypt each fragment with AES 256-bit key length as generated by the key generator for each fragment.

(c) Using *n* from chosen share policy generate the equivalent number of cloud service providers (CSPs) and initialise for storage of key shares as shown in Figure 11.

(d) Initialise 3 CSPs for encrypted fragments storage.

(e) Disperse encrypted fragments to 3 default CSPs initialised for it.

(f) Disperse key shares to the specified CSPs in accordance to *n*.

(g) Return all IDs (fragments and key shares to metadata table).

(h) Generate user's ID and file details.

(i) Input user's ID, use same to retrieve the encrypted fragments, decrypt and recover the original file as shown in Figure 12.

(j) Calculate cloud behaviours using metrics in Tables 2 and 3.

**Aim:** To measure the time taken to break each file into optimum fragments fragments, disperse and recover the original file in relation to share policy.

**Metrics:** file size over time taken to process in relation to share policy.

**Justifications:** Previous research generate the number of fragments from each file based on the number of cloud subscriptions irrespective of file size. This does not consider the effect of fragment size on the overall system overheads. This thesis, therefore, chose minimum and maximum file sizes, alongside ranges of share policies to be able to have a controlled experiment within a specific time frame. The incremental nature is to help find appropriate results that fit into the metrics being evaluated – optimum fragment size. Thus, we chose optimum fragment size from these results to enable this thesis to evaluate the scalability of our method in relation to similar methods. Using block and optimum fragment sizes at different share policies are to justify the reason for our choice at the end of the day. All files are byte streams to provide a neutral ground for all file types. Encryption method is AES 256-bit key length with Electronic Code Book (ECB), the basic method as byte streams broken into fragments were being encrypted and hence did not consider using Salt and Initialisation Vector (IV).

**Tests conducted**

**Test One: Tests of share policies, file sizes and fragments**

**Test 1a:** Varied share policies, varied file sizes, equal number of fragments and system overhead

**Test 1b:** Varied thresholds, varied file sizes, equal number of fragments and system overhead

**Test Two: Tests file sizes, fragments, and share policies**

**Test 2a:** Varied file sizes, fixed share policy, equal number of fragment and system overhead.

**Test 2b:** Varied file sizes, share policies, fixed number of fragments and system overhead.

## Resilience

**Experiment C:** Recover original file in the midst of varying rates of cloud outages

**Steps taken**:

(a) Use results of experiment B as benchmarks for normal situations in file recovery.
(b) Repeat the experiment using share policy 3 from 5 and keep file size constant.
(c) Disconnect one CSP and run the experiment as in (b) above again.
(d) Disconnect two CSPs and run the experiment as in (b) above again.
(e) Repeat the experiment using share policy 6 from 10 and keep file size constant.
(f) Disconnect three CSPs and run the experiment as in (e) above again.
(g) Disconnect four CSPs and run the experiment as in (e) above again.

**Aim:** To measure the effects of cloud outages in file recovery.

**Metrics:** File size over time taken to recover in relation to share policy.

**Justifications**: Different file sizes and varying policies will be able to give the required information on the behaviours of FSSS at different rate of cloud failures.

**Tests conducted**

**Test a:** File combination at various cloud outages against normal situations.

**Test b:** Key recovery at various cloud outages against normal situations.

## The social concept in secret sharing

**Experiment D:** Plot the average results of the metrics collected to determine the major determinant of CSP's level of cooperation during each operation.

**Steps taken:**

(a) Use metrics and standards as detailed in Tables 2 and 3.
(b) Collect each of these metrics results in a file after every operation.

(c) Plot the graphs.

(d) Observe the nature of curves and consider different behaviours of the graphs with respect to others.

(e) Categorise them as influencing and non-influencing metrics using the pattern of their curves.

**Aim:** To measure the sufficiency of using rate of availability and response time in determining every CSP's level of cooperation during each operation.

**Metrics:** As detailed in Tables 2 and 3.

**Justifications:** Taking into considerations, the results of different metrics of cloudlets during each operation will be able to give the appropriate information on the major determinants of cloud behaviours.

**Tests conducted**

**Test a:** The sufficiency of using rate of availability and response time to determine cloud level of cooperation.
**Test b:** The fairness of using capacities above to disenroll participant.

**Period of experiments:** These experiments were conducted usually from early hours in the morning through midnight each day of the experiment, sometimes earlier and later than midnight depending on the time taken to conclude each round of code executions. The reason is to ascertain behaviours of the cloud at peak and less peak hours.

### 4.4.2 Evaluation frameworks

#### Scalability

Jogalekar and Woodside [118] define scalability metric as the measure of the productivity level of a system. This thesis, therefore, defines scalability metric as the ability of a method to continue with data sharing and recovery at different file sizes and share policies. Hence, the scalability measure of FSSS will be evaluated on the ability to overcome three scalability bottlenecks experienced with the use of secret sharing scheme in data sharing and recovering in multi-cloud architectures. These are the inability of use when file size increases, the effects of share policy, which are increase in number of participants (*n*) and that of the threshold (*m*) in file creation and recovery [19], they will be considered with respect to computing resources made available such as the processing power, which includes the number of processors, RAM size, single or multithreaded and network bandwidths. All the evaluations will be on cloud-based systems.

Taking file size as $K_s$, time taken as $t_s$, Scalability is expressed as:

$$\boldsymbol{Scalability} = \frac{\boldsymbol{K_s + S_p}}{\boldsymbol{t_s}} + \boldsymbol{Cp} \qquad \text{Equation 19}$$

Share Policy *Sp* of each method will be measured as:

$Sp = \frac{n}{m}$, and where $Sp \gg 1$, it shows that $n$ has been increased, where $m$ remains unchanged, $Sp \geq 1$, *m* increased, where *n* remains unchanged.

Where S$p$ is the share policy, which is a factor of participants ($n$) and that of threshold (*m*) and *Cp* is the computing power. Whereas $K_s\,(byte), t_s\,(second)$. $Sp$ may be varied, *Cp* is meant to be an unchanging computing resources in use. Total time taken ($t_s$) is the sum of time taken in data sharing, dispersal, shares retrieval and secret recovery.

## Resilience

The ability of a system to provide an acceptable level of service despite challenges like failures has been defined as resilience according to Sterbenz *et al*. [119]. This thesis sees resilience as the ability of a method to continue to provide data availability irrespective failure rate of participants to throw in their shares for secret recovery. It is a measure of the ability of a method to avoid downtime and data loss during acceptable level of participants failures. Following these definitions, FSSS level of resilience will be evaluated on the ability to avoid downtime and data loss during cloud failures. In secret sharing scheme, Abdallah and Salleh [19] provides information that data recovery is only possible when $n \geq m$. But fails to provide further information on the accrued system overheads during each rate of failure.

Taking file size as $K_r$, time taken to recover a file during cloud failures as $t_r$, we express Resilience as:

$$\boldsymbol{Resilience} = \left(\frac{\boldsymbol{K_r + \nabla S_p}}{\boldsymbol{t_s}}\right) + \boldsymbol{Cp} \qquad \text{Equation 20}$$

Recall that in equation *19* that $Sp = \frac{n}{m}$ and here in equation 20, this thesis is dealing with only a measurement that concentrates on a depleting $n$ values due to outages at an unchanged $m$ value and hence $\nabla Sp = \frac{\nabla n}{m}$, where $\nabla$ represents decrease in value of $n$. When $Sp \geq 1$, the system shows that is nearing breaking point, whereas $Sp = 1$, the breaking point has been reached and $Sp < 1$, data recovery is impossible. $Cp$ is available computing power, assuming that all methods being evaluated have same amidst changing file size and rate of cloud failure. $K_r$ is in (byte), $t_r$ is in (second).

**For effective use of equations 19 and 20, this thesis assumes that the methods under evaluation have same computing resources as well as share policies and file sizes.**

**Measuring Scalability and Resilience**

**Scalability**

Available literature shows that the larger the $n$ in data sharing, the faster the data dispersal to participants as data shares are in smaller pieces but data sharing method has four major phases – data sharing, dispersal, retrieval and recovery. This implies that the scalability of a method cannot be judged based only on the dispersal overheads but in overall overheads of its processes. From available literature, Abdallah and Salleh [19] and [27] showed that secret sharing has inherent bottlenecks as increase in the number of participants increases data sharing overheads, while increasing the threshold increase data recovery overheads.

In order to provide a balance between these and give information on a possible combination that has less overhead, this thesis evaluation framework as shown above therefore becomes imperative. It provides a way of measuring the impact of file size, against time taken to process it given a share policy in addition to the effect of the computing resource available. To be able to evaluate different methods, it assumes that both file size, share policy and computing resources available are same. Therefore, evaluation of Scalability is the measure of accrued overheads with respect to time taken to share, disperse, retrieve and recovery secret given same file size, share policy and computing resources.

**Resilience**

Resilience measures the ability of a method to recover secret during outages. The works of Abdallah and Salleh [19] and [27] provided information supporting the evidence that when the number of shares available for recovery is less than the defined threshold, implying that $n < m$, for $n$ as the number of participants and $m$ as the defined threshold, secret recovery is impossible and hence a proof that the use of secret sharing has limited level of resilience. Based on this, the evaluation of resilience of different data sharing method is on the mesure of its ability to recover secret at different rate of outages in relation to the accrued overheads. While Scalability is a measure of accrued overheads in both data sharing, dispersal, shares retrieval and secret recovery, Resilience is concentrated on the ability to recover secret in relation to overheads accrued during shares retrieval and secret recovery during outages. So, this thesis evaluation is a measure of ability to recover secret at different rates of outages assuming that the files size, share policy, and computing resources in use remain same althrough the operations. This ability is also viewed in relation to time taken to do so.

**General results collection procedures**: In collecting results from code executions, several procedures are taken to evaluate the results to make sure all tests are coherent with none skewed in favour or against others. After each execution, cloud behavioural analyses results collected through a file are calculated automatically and displayed graphically so as to ascertain the behaviour of different metrics such as speed, throughput, download and upload bandwidths and so on (see tables 2 and 3 for details) and when they are inconsistent for more than three runs, the results are discarded and are validated otherwise with average taken.

**Table 1: User Management Data Store**

| Datetime | ID | UUID | FileName | FileSize | FileRef | Cloudlet0 | Cloudlet1 | ... | Cloudletn |
|---|---|---|---|---|---|---|---|---|---|
| Datetime$_1$ | ID$_1$ | UUID$_1$ | Filename$_1$ | Filesize$_1$ | FileRef$_1$ | FileID$_0$ | FileID$_1$ | ... | FileID$_n$ |
| .<br>.<br>. | .<br>.<br>. | .<br>.<br>. | .<br>.<br>. | .<br>.<br>. | .<br>.<br>. | .<br>.<br>. | .<br>.<br>. | .<br>.<br>. | .<br>.<br>. |
| Datetime$_n$ | ID$_n$ | UUID$_n$ | Filename$_n$ | Filesize$_n$ | FileRef$_n$ | FileID$_{n0}$ | FileID$_{n1}$ | ... | FileID$_n$ |

Legends:

| | |
|---|---|
| DateTime | = specifies Date and Time user details were stored. |
| ID | = contains hashed value of all user personal details |

| | |
|---|---|
| UUID | = Universally Unique Identifier automatically created for each user. It differentiates users and used to name file fragments when created in order of creation (UUID.1, UUID.2,..,UUID.n). |
| FileName | = specified name of file from which fragments were created |
| FileSize | = provides size of user's file |
| FileRef | = Reference number of user's file, serves as a Primary key. |
| Cloudlet0-Cloudletn | = provides FileID that reveals share storage details (cloud API, token, name of share and location in storage bucket of the cloud). |

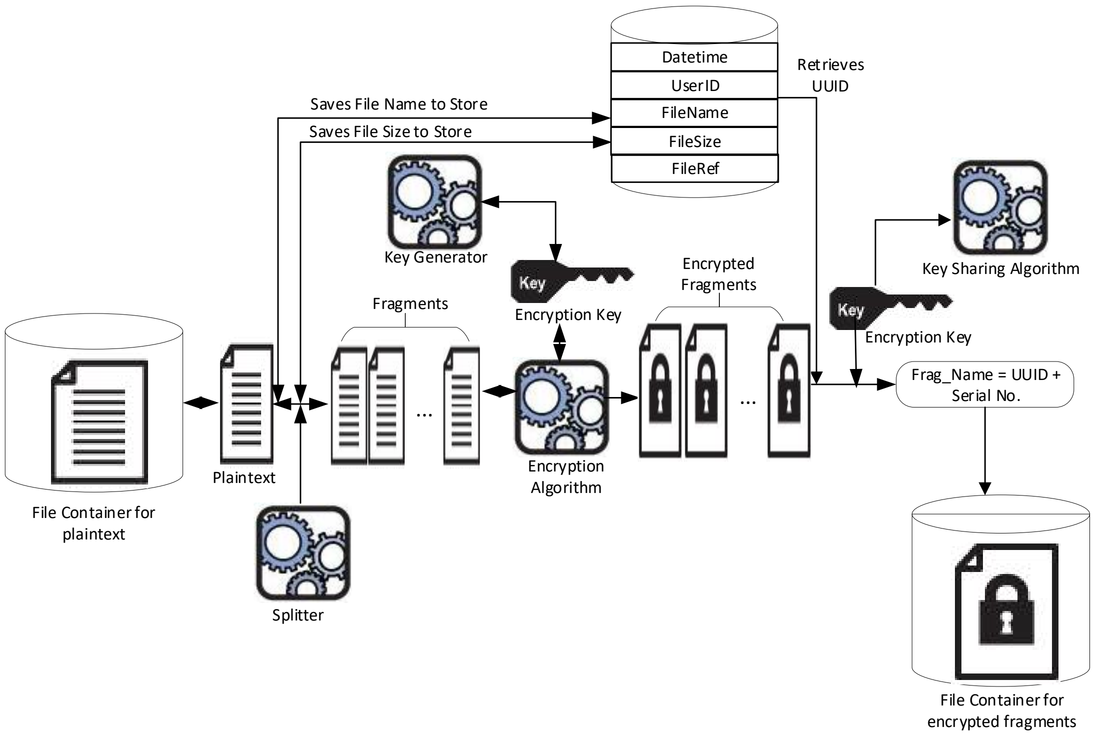


Figure 10: File Fragmentation and Encryption

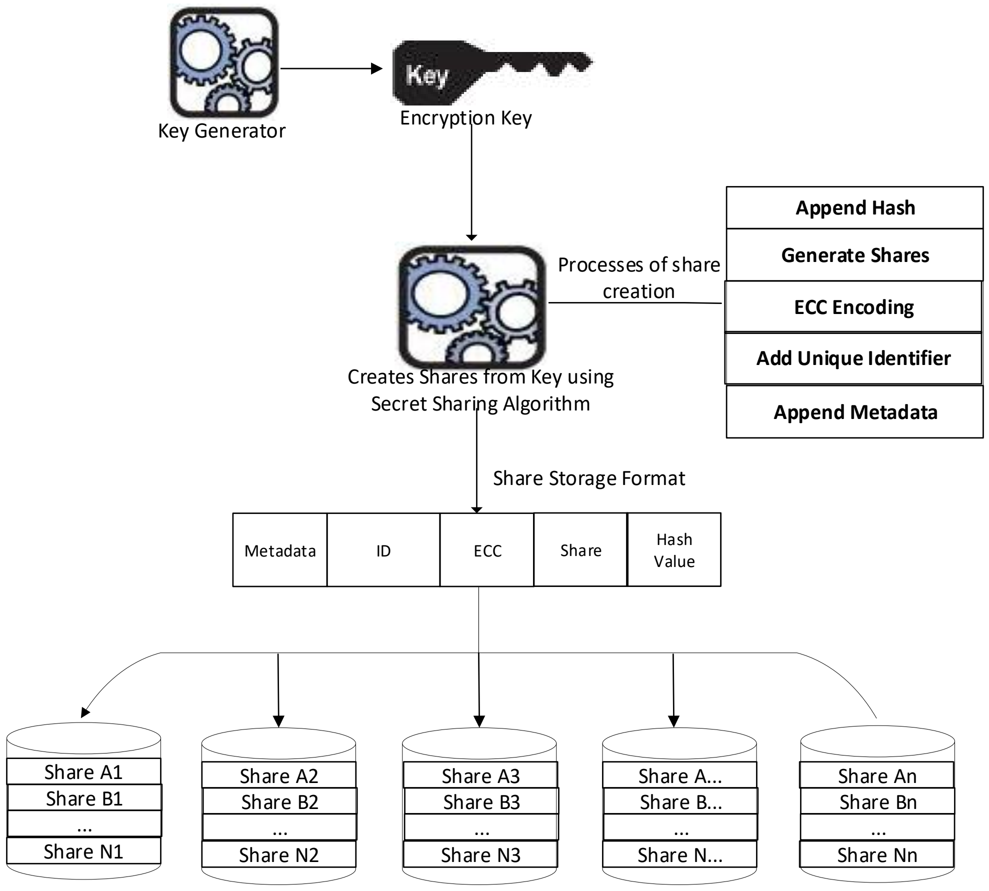


Figure 11: Key Share Storage

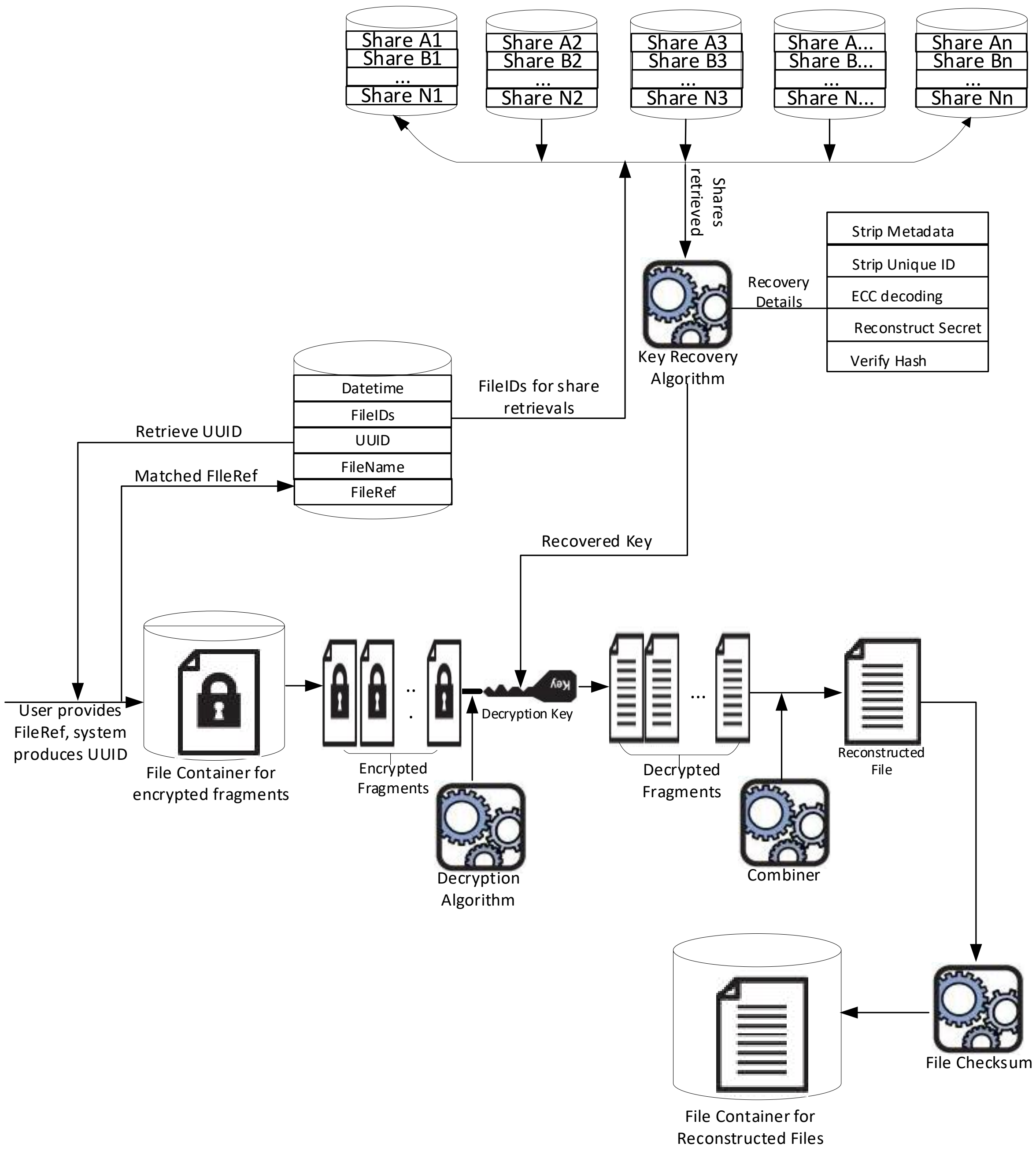


Figure 12: Key recoveries and file reconstruction

## I. Capacity Measures of clouds

Table 2 shows the Capacities, Evaluation Metrics, Formulae, Units and Tools for measurement.

### Justifications

In order to measure cloud capacities, we relied on the first attempt to benchmark this type of study as was conducted by Li et al. [120]. This thesis relied on this as it a comprehensive catalogue of metrics for evaluating commercial cloud services and this follows similar works in Bardsiri and S. M. Hashemi [121] on – "Qos metrics for cloud computing services evaluation".

Table 2: Evaluation Metrics

| Capacity | Metrics | Formulae | Unit | Tool |
|---|---|---|---|---|
| Availability | Packet Loss Frequency | Packet Loss Per Unit Time | Bits per second (bps) | Pingparser |
| Latency | IP Transfer Delays | Propagation Delay + Serialisation delay | Milliseconds (ms) | SpeedNet |
| Transaction Speed | Max. No. of Transfer Session | Length of File Over Time | Meters per Second (mps) | SpeedNet |
| Throughput | Volume of processed data | File Size Over Time | Ops/Sec (bps) | SpeedNet |

## II. Overall Performance Measures

Overall performance extends some measures by including more metrics that shows comparatively cloudlets performance at a glance by considering their different capacities such as: Elapsed Time (s), Packet Loss, Average Round Trip Time (ms), Speed (s), Download and Upload Bandwidths (bps), Latency (bps) and Throughput (bps).

Table 3: Overall Performance Evaluation Metrics

| Capacity | Metrics | Formulae | Unit | Tool |
|---|---|---|---|---|
| Elapsed Time | Time Taken for action | End Time - Start Time | seconds | Clock Time |
| Packet Loss | Packet differentials | Received – Transmitted packets | bps | SpeedNet |
| Average Round Trip Time | Mean Round Trip Time | Total RTT/Number of Round Trips | ms | Pingparser |
| Speed | Max. No. of Transfer Session | Length/Time taken | mps | Pingparser |
| Download Bandwidth | Total download traffic carrying capacity | Volume of data transmitted between two points per second | bps | SpeedNet |
| Upload Bandwidth | Total upload traffic carrying capacity | Volume of data transmitted between two points per second | bps | SpeedNet |
| Latency | IP Transfer Delays | Propagation Delay + Serialisation delay | ms | SpeedNet |
| Throughput | Volume of Processed Data | File Size Over Time | bps | SpeedNet |

## 4.5 System Specifications

System specification deals with the specifics on cloud infrastructure used. It takes into accounts the hardware and software components.

### 4.5.1 Virtual Machine Specifications

Standard Intel N1, 1 vCPU 2.30GHz Intel Zeon (R), 3.840GB memory, 50.00GB NTFS HDD (Boot disk), Windows Server 2012 R2 DataCenter edition, Stand-alone, Terminal Server, Ethernet, Red Hat VirtIO Ethernet Adapter on Google Cloud Console running on Network Internet Egress from Americas to Americas.

Beside the VM, several cloud storage facilities were used to stored data such as Google Cloud, Amazon Web Services Simple Storage Service (AWS S3), Dropbox and Microsoft OneDrive. AWS S3 has the highest number of storage subscriptions differentiated by their different regions such as shown in Table 4.

**Table 4: Cloud Storage Locations**

| S/No. | Bucket Name | Region/Location |
|---|---|---|
| 1 | awsbucketuseast | US East (Ohio) |
| 2 | awsbucketuswest | US West (N. California) |
| 3 | awsbucketasia | Asia Pacific (Mumbai) |
| 4 | awsbucketcanada | Canada (Central) |
| 5 | awsbucketlondon | EU (London) |
| 6 | awsbucketsaopaulo | South America (Sao Paulo) |
| 7 | awsbucketfrankfurt | EU (Frankfurt) |
| 8 | awsbuckettokyo | Asia Pacific (Tokyo) |
| 9 | Dropbox I & II | US West of Northern California) |
| 10 | Microsoft OneDrive | Unknown |

## 4.6 Conclusions

This thesis has been able to present using this chapter the nature of FSSS's design by providing details on the components that make up the system. It includes steps used in designing the system through coding, execution of codes, result collections and computations to determine cloud behaviours during process executions. With the design principles explored, information has been laid in a nutshell what this thesis tends to achieve and how it goes about it. Furthermore, the System design and components presented all the subsystems that make up the system such as - user management, file fragmentation, recreation, cloud behavioural computations and agent which forms our future works. While the user management deals with user information collection, storage and retrievals, file fragmentation concentrates on how fragments are created, key generation, encryption of fragments and share creation. Finally, the methodology laid out two important issues bothering on experimental setup and evaluation framework that will guide the evaluation works in Chapters 5, 6 and 7, which provide details on how the objectives of the thesis were achieved and all questions answered. It ended with system specifications.

# 5 Data Fragmentation Evaluation

## 5.1 Introduction

Several methods exist in data fragmentation and have become popular with the advent of multi-cloud architecture in data storage as a way of improving data availability through redundancy technique. Popular among them is secret sharing scheme and of utmost importance of this method, aside from data availability, is the keyless encryption technique that comes with it, which gives data adequate protection as data exist in meaningless format. Nevertheless, of all these advantages, the issue of scalability remains a challenge facing its applications in large-scale data infrastructure making room for more research inputs in this direction. This thesis therefore uses this chapter to evaluate a method of fragmentation in Fragmented Secret Share System (FSSS) alongside similar schemes that are built on secret sharing scheme to measure their levels of scalability in a multi-cloud environment.

FSSS within the limit of data sizes in use in these experiments takes 15% of file size as an optimum fragment size and uses same to break file into fragments, encrypts each fragment with AES 256-bit key length, and using secret sharing scheme provide key management for each of the encryption keys. The evaluation of this method against some existing ones that apply secret sharing scheme in data sharing and recovering will not only concentrate on their scalability, but on fitness for use in large-scale data infrastructure with regards to their accrued storage overheads and these will be done using the defined evaluation framework on scalability in Section 4.4.2.

## 5.2 Data fragmentation schemes

In storage technology, data sharing is used to break data into shares and subsequently dispersed to multiple storage locations and original data is recovered when needed sometimes with a fewer number of dispersed data known as a threshold. There are many types of data sharing techniques, the foremost being the works of Adi Shamir [2] and George Blakeley [3] later to be known as Secret sharing scheme, while the scheme provides the method of securing data without encryption keys in a dispersed storage, it has proved to be insufficient for use with large-scale data infrastructure as they are not very scalable according to Buchanan [39], hence the use in securing encryption keys mainly as it will be too processor intense to use in securing data [122], [39]. Shamir and Blakely's works came to be known as perfect secret sharing scheme as they provide perfect secrecy to data because the size of each share is equivalent to that of the secret implying that their entropy is *1*. Other variants of data fragmentation of interest to us are Information Dispersal Algorithm (IDA) by Michael Rabin [40], which tends to reduce the storage complexity experienced in Adi Shamir's Perfect secret sharing scheme (PSS). Our interest is in its wider application for data dispersals since it has high-performance throughput when used to disperse data whether in-memory or in the cloud, as it does not encourage redundancy with a trade-off on the perfect security of data.

Hugo Krawczyk [11] proposed the Computational Secret Sharing (CSS) technique (a.k.a. *secret sharing made short*), which combines Rabin's IDA with Shamir's PSS. Data is first encrypted with a randomly generated key, using a symmetric encryption algorithm. Next, this data is split into *n* fragments using Rabin's IDA with a threshold *t* configured. In this case, the scheme is *t* times more efficient than Shamir's PSS. The final step is to use Shamir's PSS to produce shares of the randomly generated symmetric key (which is typically of the order of 64 to 256 bits) and then give one share and one fragment to each shareholder.

A related approach, known as AONT-RS [52] as supported by [18], applies an All-Or-Nothing Transform (AONT) to the data as a pre-processing step to the IDA. AONT guarantees that any number of shares less than the threshold is insufficient to decrypt the data. It combines AONT with Reed Solomon (RS) in order to achieve high security in a reduced computational and storage overheads. Kapusta *et al.* [18], and Kapusta & Memmi [123], developed a method that situates between Hugo Krawczyk's CSS and Rabin's IDA methods in developing a scheme that they claimed provides a lightweight data fragmentation scheme with good space efficiency and computational level of data confidentiality for data protection with application in multi-cloud environment.

### 5.2.1 Application Scenarios

Some fine-grained research projects implement the use of data fragmentation in their design include, but not limited to, Loruenser *et al.* [25] presented an architecture for secure cloud-based data sharing known as ARCHISTAR based on secret sharing scheme. The focus of the system is on providing adequate confidentiality to data; make it available against any active attacks as well as robust even in the face of failures.

Ermakova and Fabian [13] defined a secret sharing for health data in multi-provider clouds. Their work was based on the need to provide a scheme that will make data readily available, provide confidentiality and integrity to medical records stored in clouds. They used a secret sharing scheme to distribute data as fragments to several clouds in order to provide the needs as stated above.

There are other research solutions based on secret sharing schemes in multi-cloud environment such as, Ukwandu *et al*. [46], presented RESCUE: Resilient Secret Sharing Cloud-based Architecture that defined an architecture that applied the resilient nature of secret sharing scheme in building a robust data sharing in multi-cloud environment. Alsolami and Boult [17], worked on CloudStash: Using Secret-Sharing Scheme to Secure Data, Not Keys, in Multi-Clouds. The works used secret sharing for data sharing in multi-cloud environment and Fabian *et al.* [12] on collaborative and secure sharing of healthcare data in multi-clouds that applied the use of secret sharing in sharing and recovery of the file in conjunction with cryptographic primitives.

## 5.3 FSSS Evaluations

The method presented in this thesis is known as fragmented secret share system. It creates fragments from a file, encrypts each fragment with different encryption key and applies secret sharing methods as used in cryptography to create robust and secure cloud-based keyless key management system using multi-clouds architecture for storage management.

### 5.3.1 Method of Data Fragmentation

The experiments focused on providing information on choices that will ensure quick, efficient, and scalable operations through a proper combination of fragments, file sizes, and share policies. It provides information on a relationship between file and fragment size and the appropriate policy combinations at different users' choice of file size and share policy.

Experiments performed are of two variants, A and B as presented earlier in Section 4.4.1. All files are byte streams generated and stored as that provides an opportunity of testing in a neutral environment suitable for all file types. In all, experimental evaluations will be based on results obtained from file fragmentations and key shares. The plots will be on varied file sizes in kilobytes (KB) against time taken in milliseconds (sec) using varied key sharing and recovering policies.

The total overhead cost of each file processed is calculated by summing all times taken to break the file into fragments; encrypt all accrued fragments; decrypt all fragments and combination of the file.

#### 5.3.1.1 Fragments Storage

FSSS data mappings techniques as shown in Figure 13 has to do with how encrypted fragments of file are mapped into storage buckets in the cloudlets with the fileIDs (file and fragments identification details) containing all the details of the fragments, including the cloud API tokens, fragments locations, names and serial numbers returned to a metadata server from where each is retrieved and used to reconstruct the original file when needed.

The mappings are used for both fragments in three separate storage buckets made up of premised and external storages in Google Cloud (premised), AWS S3 (external public cloud) and Dropbox (external private cloud). The value *n* represents the last fragment. The mixture of public and private clouds premised and external were intended to provide the needed resilience and ease of access hence improve performance by reducing overhead costs to file reconstruction. The Google cloud storage (premised) is the cloud in use daily, while access to others are mainly when there is an outage in Google cloud or an evidence of a corrupt fragment(s) during file reconstruction. This method uses redundancy technique to improve data availability but fails to implement a threshold scheme thereby making it impossible to recover the original file if one of the fragments is lost, corrupted or fails during retrieval from different storage buckets. This drawback results to a retrieval of fragments using an alternative storage bucket and hence increases the recovery overheads at this point.

### 5.3.2 Method of Key Sharing and Recovering

The base algorithms – Sharing and Recovering are the concept as presented originally by Adi Shamir [2] and hence a perfect secret sharing scheme. In experiment A, the overhead cost is the sum of time taken to create key shares; write key shares to storage devices; recover key shares from storage devices based on defined threshold for key recovery. The time taken are quite infinitesimal but a reference to them are necessary for comparison with that of experiment B, which was done using different cloud service providers presenting a real-life situation to share creation; share writing; share recovery based on a prevailing threshold and key recovery. In experiment B the overhead cost for key sharing and recovery are based on time taken in key sharing, share writing to cloudlets which is made up of upload and download times, share recovering from downloads and secret key recovering.

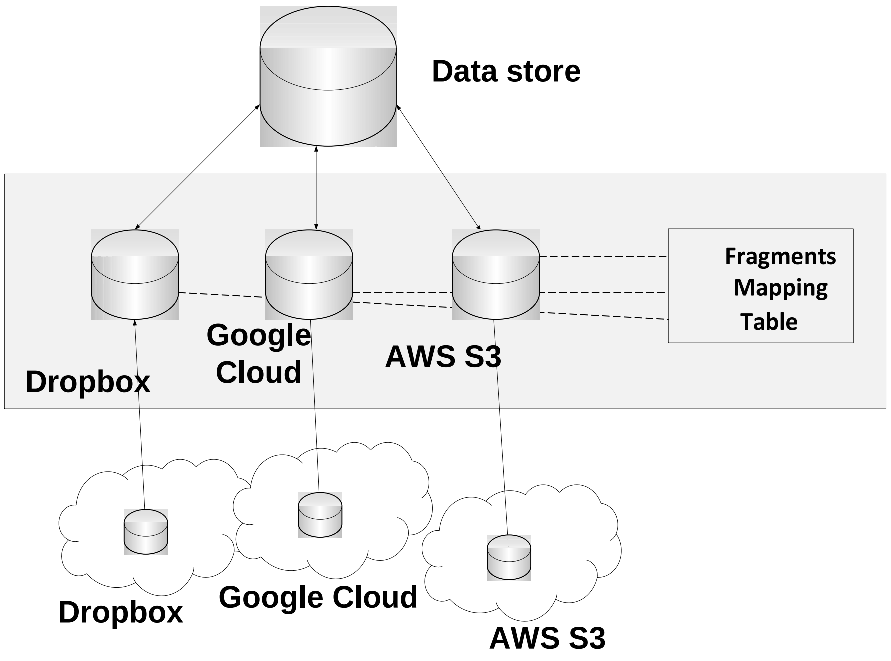


Figure 13: FSSS Encrypted Fragments Mappings to Cloudlets

#### 5.3.2.1 Key Shares Mapping

FSSS key shares mapping as shown in Figure 14 below has to do with how key shares of encryption key used in safeguarding fragments of file are mapped into storage buckets in the cloudlets with the shareIDs (share identification details) containing all the details of the shares, including the cloud API tokens, share locations, names and serial numbers returned to a metadata server from where each is retrieved and used to reconstruct the decryption keys when needed. The mappings are done using a maximum of ten different Cloud storages mainly AWS S3 in conjunction with Dropbox and Microsoft OneDrive. The mixture of public and private clouds was intended to provide the needed resilience and ease of access hence improve performance by reducing overhead costs to key recovery. Just like every other Shamir PSS method implementation, this method suffers from storage complexity as each share size is equivalent to secret key size and hence the larger the key size, the more complex the storage and system overheads. But suffice it to say that the choice of threshold (*m*) in relation to the total number of participating clouds (*n*) impacts on the overall system overheads, this thesis therefore sees the choice of *m* within the context and limit of computer resources, file sizes and share policies used in the experiments as best around 60% of *n* for an optimum performance. This may vary depending on the computing resource in use.

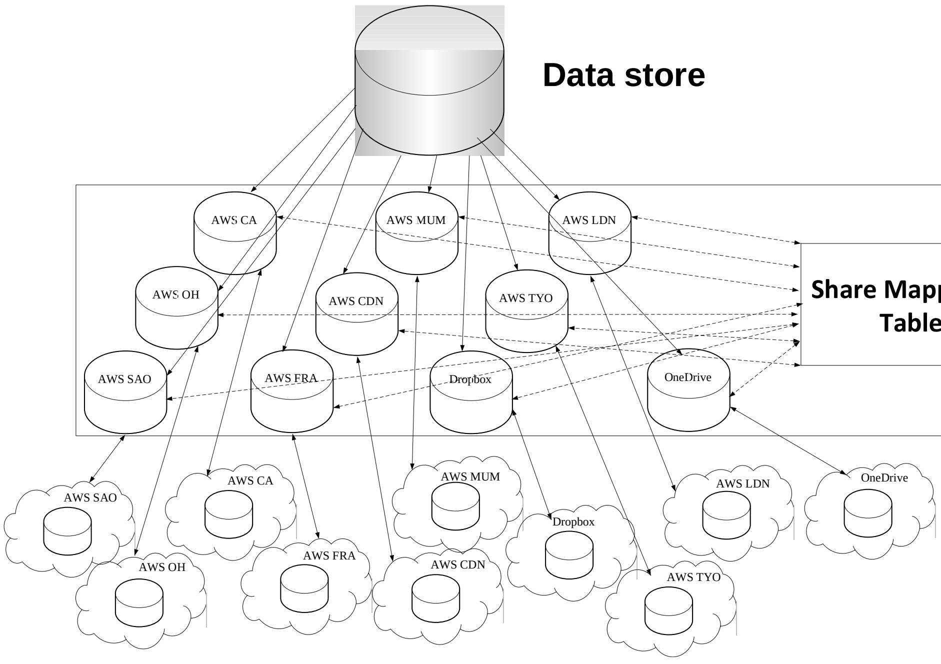


Figure 14: Key Shares Mappings to Cloudlets

## 5.4 Tests Results and Evaluations

#### 5.4.1.1 Test One

In the first evaluation (see Table 5), the best combination of the system that provides a high level of data scalability is evaluated. This assumes that data fragmentation irrespective of fragment size and share policy is scalable.

This test showed that file fragmentation using varied fragment size is scalable but has higher overhead when fragment size is smaller in relation to file size as seen in Table 5 and graphically in Figure 15. The evidence here is such that the smaller the fragment size in relation to file size, the higher the number of fragments generated thus an increase in overhead as seen in 1KB, 2 from 5 below, where the system failed to combine a file of 1GB size using 1KB fragment size due to large system overheads. In 1KB, 3 from 5 and 1KB, 4 from 5 provided evidence that increasing the threshold with smaller fragment size, will lead to high system overhead, implying the effects of threshold and fragment size on system performance, thus dismissing the above assumption.

#### 5.4.1.2 Test Two

In the second test, we evaluated the best combination of the system that provides data scalability with less overhead. This assumes that data fragmentation using an equal number of fragments irrespective of share policy is highly scalable.

The above assumption is supported by the evidence from Table 6 and Figure 16 in comparison with Table 5, thus providing a proof that file fragmentation is highly scalable when fragment size is defined as 15% percentage of file size. Though cautions should be applied in giving all file sizes same percentage of fragment size as evidence showed that the larger the file size, the more fragment percentage will be.

#### 5.4.1.3 Test Three

This test assumes that share policy has no effect in file sharing and recovery. Using Table 7, this thesis validates the earlier claim by Abdallah and Salleh [19] that increasing the threshold and that of participants increases system overhead during share creation and secret recovery. But Table 8 states otherwise and proves that FSSS as a method improves on these bottlenecks by combining data fragmentation with secret sharing using optimum fragment size.

#### 5.4.1.4 Test Four

This test assumes that FSSS will be adversely affected and fails to produce results when file size increases as well as change in share policy while keeping the computing resources constant.

Table 8 above shows that FSSS continued producing results as share policy changed. Both the increase in threshold and that of number of participants did not show significant effects indicating that it will not fail to provide result at different application scenarios. It also did not show any significant effects as file sizes increased exponentially.

**Units of measurements**

---

Units of measurements are (B) for bytes used in measuring file sizes, and (S), for seconds used in measuring time taken to perform a task using a certain share policy.

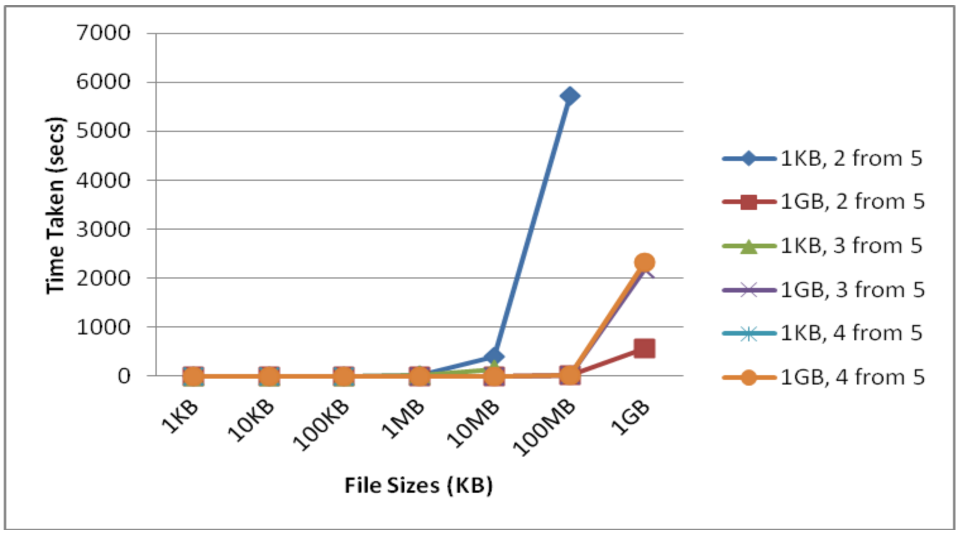


Figure 15: Measuring scalability using varied fragment sizes, share policies and file sizes.

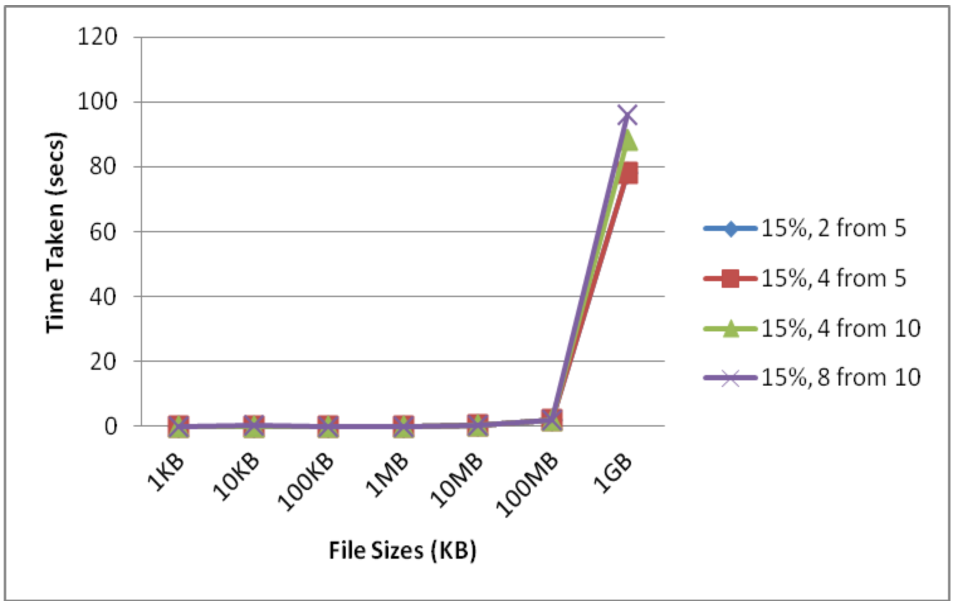


Figure 16: Plot of varied file sizes, share policies and equal number of fragments.

Table 5 : Measuring scalability using varied fragments sizes, share policies and file sizes

| S/N | FileSize (B) | 1KB, 2 from 5 (S) | 1GB, 2 from 5 (S) | 1KB, 3 from 5 (S) | 1GB, 3 from 5 (S) | 1KB, 4 from 5 (S) | 1GB, 4 from 5 (S) |
|---|---|---|---|---|---|---|---|
| **1** | 1KB | 0.06 | 0.06 | 0.05 | 0.06 | 0.13 | 0.20 |
| **2** | 10KB | 0.79 | 0.02 | 0.29 | 0.08 | 0.40 | 0.08 |
| **3** | 100KB | 1.41 | 0.0.6 | 1.38 | 0.07 | 2.36 | 0.16 |
| **4** | 1MB | 13.06 | 0.09 | 10.58 | 0.10 | NC | 0.23 |
| **5** | 10MB | 403.75 | 0.59 | 143.50 | 0.48 | NC | 0.51 |
| **6** | 100MB | 5716.87 | 8.11 | NC | 9.26 | NC | 10.41 |
| **7** | 1GB | NC | 572.66 | NC | 2168.81 | NC | 2316.12 |

Table 6: Measuring scalability with varied file sizes, share policies and equal number of fragments

| S/N | FileSize (B) | 15%, 2 from 5 (S) | 15%, 4 from 5 (S) | 15%, 4 from 10 (S) | 15%, 8 from 10 (S) |
|---|---|---|---|---|---|
| **1** | 1KB | 0.02 | 0.02 | 0.07 | 0.05 |
| **2** | 10KB | 0.08 | 0.02 | 0.04 | 0.12 |
| **3** | 100KB | 0.05 | 0.02 | 0.06 | 0.04 |
| **4** | 1MB | 0.04 | 0.03 | 0.06 | 0.10 |
| **5** | 10MB | 0.18 | 0.17 | 0.21 | 0.26 |
| **6** | 100MB | 1.87 | 1.74 | 1.76 | 1.78 |
| **7** | 1GB | 78.10 | 78.01 | 88.33 | 96.10 |

Table 7: Measuring the effects of policies, number of fragments generated on key recovery.

| S/N | FileSize (B) | 1KB, 2from5 (S) | 1KB, 3from5 (S) | 1KB, 4from5 (S) | 15%, 2from5 (S) | 15%, 4from5 (S) | 15%, 4frm10 (S) | 15%, 8frm10 (S) |
|---|---|---|---|---|---|---|---|---|
| **1** | 1KB | 0.017 | 0.0184 | 0.039 | 7.094 | 8.160 | 18.675 | 1536.350 |
| **2** | 10KB | 0.122 | 0.130 | 0.379 | 7.254 | 7.397 | 16.971 | 1629.540 |
| **3** | 100KB | 1.050 | 1.566 | 99.376 | 7.385 | 6.918 | 17.866 | 1484.630 |
| **4** | 1MB | 12.606 | 53.050 | NC | 7.670 | 6.793 | 18.672 | 1444.335 |
| **5** | 10MB | 331.684 | 4221.852 | NC | 8.456 | 8.236 | 16.395 | 1515.853 |
| **6** | 100MB | 3839.531 | NC | NC | 7.290 | 6.655 | 15.920 | 1472.754 |
| **7** | 1GB | NC | NC | NC | 6.768 | 7.518 | 16.156 | 1599.017 |

Table 8: Overhead Cost of File Combination using equal number of fragments with varied share policies

| S/N | FileSize (B) | 15%, 2 from 5 (S) | 15%, 3 from 5 (S) | 15%, 4 from 5 (S) | 15%, 4 from 10 (S) | 15%, 6 from 10 (S) | 15%, 8 from 10 (S) |
|---|---|---|---|---|---|---|---|
| **1** | 1KB | 0.02 | 0.03 | 0.02 | 0.07 | 0.02 | 0.05 |
| **2** | 10KB | 0.08 | 0.03 | 0.02 | 0.04 | 0.02 | 0.12 |
| **3** | 100KB | 0.05 | 0.03 | 0.02 | 0.06 | 0.03 | 0.04 |
| **4** | 1MB | 0.04 | 0.04 | 0.03 | 0.06 | 0.04 | 0.11 |
| **5** | 10MB | 0.18 | 0.19 | 0.17 | 0.21 | 0.17 | 0.26 |
| **6** | 100MB | 1.87 | 1.59 | 1.74 | 1.76 | 1.50 | 1.78 |
| **7** | 1GB | 78.10 | 81.38 | 78.01 | 88.34 | 79.98 | 96.10 |

**Legend:**

**NC = Not Computable within the limit of computing resources used in these experiments.**

## 5.5 Overall Evaluation of Data Fragmentation Methods

Secret sharing scheme has been identified as primarily for secure key management as posited by Narani in [122] and this position is supported by Buchanan [39], Buchanan *et al.* [31], Kapusta *et al* [18], Abdallah and Salleh [19], Koikara *et al* [20], and Pal *et al.* [21]. But Alsolami and Boult in [17] opposed this by saying that Secret Sharing is for Data Security, not Keys. Works of Ermakova and Fabian [13] and Fabian *et al* [12] are in conformity with theirs as they used combination of secret sharing scheme with other cryptographic primitives in securing healthcare data through shares creation, dispersal and recovery in multi-clouds.

This thesis evaluation at this point will focus on FSSS in comparison with that of Fabian *et al.* and Alsolami & Boult using the developed evaluation framework on scalability in Section 4.4.2. The reason for our choice is that these two experiments were conducted using cloud resources and showed some levels of scalability as against others that were conducted using internal hard disk storage for share storage and retrievals. In the same vein, the work of Shor *et al.* [4], will not be evaluated here as its use of data fragmentation technique is in key management, which is secret share system that already being discussed.

FSSS was conducted on Standard Intel N1 1 vCPU 2.30GHz Intel Zeon (R), 3.840GB memory, 50.00GB NTFS HDD (Boot disk), Windows Server 2012 R2 DataCenter edition, Stand-alone, Terminal Server, Ethernet, Red Hat VirtIO Ethernet Adapter on Google Cloud Console running on Network Internet Egress from Americas to Americas with 13 different cloud storage buckets, 3 for data while 10 for keys storages, majorly AWS S3 storage buckets in conjunction with Microsoft OneDrive, Google Cloud and Dropbox, while that of Fabian *et al* conducted theirs using Windows 7 Professional 64-bit machine within Oracle JRE 1.6.0_39 and Java HotSpot 64-bit Server VM. Intel Core i5 2500 K, on 4x4841 MHz, 8 GBytes DDR3 RAM, Dual Channel on 686,9 MHz with AWS S3 as storage buckets and all operations were executed using multithreading, while Alsolami & Boult used machines that run on Intel core i5 CPU 2.40 GHz, 4GB RAM and 64 bit Linux Operating System with AWS S3 as storage buckets.

The scalability of these methods will be evaluated based on the ability to continue production as file size increases. It will also take a look at the effects of share policy on system overheads. Table 8 shows that FSSS was not affected significantly by an exponential increase in file sizes, an indication that it will not fail to share and recover file at any change of file size and share policy. These are indications that FSSS is scalable. Table 9 is an overall summary of the behaviours of the works of Fabian *et al* and CloudStash at different file sizes and share policies.

In Table 9, file increase implies the method is affected by file size increase, while policy change implying that increasing the number of participants and that of threshold significantly affect share creation and file recovery. As earlier stated this thesis sees scalability from the perspective of continuous production as file size increases. It also based on the ability of a method to overcome the effect of increase in the number of participants and that of threshold in share creation and secret recovery. In all FSSS showed evidence of improving the process of data fragmentation in conjunction with secret sharing scheme by reducing the adverse effects of file size increase and that of share policy change in file recovery as shown in Table 8. Table 8 also shows that it can continue production as file size increases and share policy changes. These therefore, prove that FSSS is more scalable than the two methods as evaluated as shown in Table 9.

Table 9: Evaluation table for FSSS and other similar methods to measure scalability

| S/N | Project Name | Method Used | Storage Device | Maximum File Size | File Increase | Policy Change | Processor | Scalability |
|---|---|---|---|---|---|---|---|---|
| 1 | CloudStash | SSS | Cloud | 10MB | Yes | Yes | Single Thread | Poor |
| 2 | Fabian *et al* | SSS | Cloud | 500MB | Yes | Yes | Multi-Thread | Average |
| 3 | FSSS | FSSS | Cloud | 1GB | No | No | Single Thread | Good |

## 5.6 Conclusions

In current data sharing and recovery scenarios, several methods exist and many of the methods applied the use of Secret sharing scheme alongside other cryptographic primitives such as symmetric encryption like AES with the opinion that secret sharing scheme is only good for securing cryptographic keys [123], [18], [19] and others, while [124], [17], [12] and [13] are of the opinion that secret sharing scheme should be directly applied in data sharing and recovering as it is resilience and self-preserving in application. FSSS was borne after our previous works in [31] and [46] and as was initially evaluated in [5]. It shares the view of the former and hence limited itself in evaluating the method against those that shared the second view so as to add its voice on the need for high level scalability in current data sharing and recovery mechanisms that are fit for use in large-scale data infrastructure while maintaining the resilient, secure and self-preserving nature of data fragmentation schemes. The method is not devoid of the secured nature of data fragmentation schemes, it rather improved on it by providing a four-layered protection to data using data fragmentation, symmetric encryption, secret sharing scheme and redundancies. Above all it is scalable and fits into use for sharing and recovering of large-scale data irrespective of size and types despite the shortcomings as identified in the above evaluations. Finally, this chapter concludes application scenarios of data fragmentation in a multi-cloud environment to show how previous methods have been applied in multi-cloud environment.

# 6 Key Management Evaluation

## 6.1 Introduction

The previous chapter was used to provide FSSS's data fragmentation scheme as well as evaluate same with current practise that believes that secret sharing scheme is scalable. In this chapter, FSSS key management is presented with a view of evaluating same with similar methods that uses secret sharing scheme in key management in the cloud as FSSS combines data fragmentation with encryption and manages the keys using secret sharing scheme. Three things are obvious in this chapter, one is that FSSS disagrees with Nojoumian *et al.* that the use of rate of availability and response time is enough to judge participant's (cloud) level of cooperation during each operation, two is that it validates the facts that data recovery is possible in as much as cloud failure (*n*), is greater than or equal to the defined threshold (*m*) and finally is to provide information on the effects of different rates of cloud failure (*n*) on file recovery.

## 6.2 Cloud-based Key Management

Security of stored data in cloud has been opined to be very crucial in cloud computing, Wang *et al.* [94] posits that such necessitates the need for the design of a key management scheme that is reliable for safe computing in the cloud. Rao [95] agrees that key management is not standardised optimally in the cloud. Rao and Selvamani [96] are of the view that having only authorised users to have access to decryption key is the best management. While Zissis and Lekkas [97] suggest that having a trusted third party is the way to go. In order to achieve this objective as posited, two major methods suffice – keyless and In-house key storage management system. In keyless system [18], [123], [25], [125] and [78] used secret sharing scheme, but Resch and Plank [52] dispersed encryption key with the encrypted data in cloudlets, while [87], [98] proposed key aggregate key management system in managing In-house key storage. Shor *et al.* [4] gave credence that use of an efficient secret sharing scheme to secure encryption key in multi-cloud environments is an optimal method of safeguarding encrypted data. In all it shows how important key management is in cloud-based data storage. From the methods presented, keyless system seems to provide a system that prevents key loss, theft and leakages and as such is resilient, reliable and as well provide confidentiality and availability needed in key management system for a robust cloud-based storage.

## 6.3 FSSS Key Management

FSSS key management is a distributed multi-cloud keyless system that is patterned after the works of Nojoumian *et al.* [32], [33] and [34] known as Social secret sharing scheme with a redefinition of what constitutes good 'behaviour' amongst participating cloudlets. While Nojoumian *et al* defines a three-fold construction denoted by (*Sha, Tun, Rec*) implying secret sharing, social tuning and secret recovery, where social tuning is carried out based on a value derived through a calculated trust function using participants' response time and rate of availability over a cumulative period. This thesis believes that such value derived is insufficient to judge participants' 'behaviour' by proving that cloud behaviours such as response time is subjective as multi-tenant nature of cloud resources make it fluctuate and hence alter values based on the amount of threads available to be processed within each period. Thus argues the sufficiency of using availability rate and response time to determine cloud level of cooperation as against literature like [120], [121], [126], [127], [128], [129], and [130] and the fairness of using capacities that show current behaviours to disenroll participants as well as adjust shares of 'cooperative' clouds without taking a look at future predictive behaviours that will help generate large volume of data thereby take cautious measures before such actions are carried out.

### 6.3.1 Tests Results and Evaluations

#### 6.3.1.1 Test One

These tests the sufficiency of using availability rate and response time to determine cloud level of cooperation. It assumes that using cloud capacities like availability rate and response time are sufficient to measure cloud participant's level of cooperation or behaviour during data sharing and recovery operations.

**Table 10: Cloud capacities measure using different CSPs (Shares Download from 4 from 10 share policy)**

| | AWS California | AWS Canada | Dropbox | AWS Frankfurt | AWS London | AWS Mumbai | OneDrive | AWS Ohio | AWS Sao Paulo | AWS Tokyo |
|---|---|---|---|---|---|---|---|---|---|---|
| Download Time (s) | 0.41 | 0.37 | 0.72 | 0.78 | 0.80 | 1.54 | 0.90 | 0.70 | 0.10 | 1.24 |
| Latency (ms) | 49.04 | 48.97 | 49.18 | 49.29 | 49.08 | 49.03 | 49.13 | 49.04 | 49.29 | 49.15 |
| Throughput (bps) | 3981.17 | 4369.51 | 2301.20 | 2101.72 | 2101.67 | 1070.80 | 1952.58 | 3191.37 | 1646.11 | 1318.31 |
| Speed (mps) | 1673169.79 | 2208192.69 | 1563442.77 | 2406985.33 | 1967461.57 | 2023129.47 | 13909024.69 | 2012561.41 | 1778988.55 | 1872583.77 |
| RTT_Avg (ms) | 27 | 27 | 10 | 27 | 27 | 28 | 10 | 27 | 27 | 27.5 |
| Availability (bps) | 0.99 | 0.99 | 0.99 | 0.99 | 0.99 | 0.99 | 0.99 | 0.99 | 0.99 | 0.99 |
| Upload Bandwidth (mbps) | 124.19 | 132.60 | 130.06 | 124.78 | 108.15 | 120.16 | 129.09 | 112.29 | 127.81 | 133.87 |
| Download Bandwidth (mbps) | 174.40 | 81.39 | 91.46 | 274.11 | 59.13 | 198.02 | 171.84 | 316.41 | 212.08 | 121.80 |
| File Size (B) | 1GB | 1GB | 1GB | 1GB | 1GB | 1GB | 1GB | 1GB | 1GB | 1GB |
| Share Size (B) | 1638B | 1638B | 1638B | 1638B | 1638B | 1638B | 1638B | 1638B | 1638B | 1638B |
| Period of experiment | 19:14:18 – 19:46:05 | 19:15:15 - 19:47:01 | 19:12:23 – 19:44:12 | 19:16:37 – 19:48:28 | 19:15:43 – 19:47:28 | 19:14:48 – 19:46:33 | 19:12:50 – 19:44:40 | 19:13:50 – 19:45:35 | 19:16:11 – 19:47:59 | 19:13:24 – 19:47:07 |

Table 10 provides some measures taken when recovering a 1GB file. First step is to retrieve all encrypted fragments made from the file, decrypt and then recombined to recover the original file. To do this, as each encrypted fragment is retrieved, appropriate key shares are retrieved from 10 participating CSPs but uses the defined threshold to recover the original symmetric key. This recoverd key is used to decrypt the encrypted fragment. This process is carried out for each encrypted fragment. All decrypted fragments are combined to recover the original file. The share size is 1638 bytes on a 4 from 10 share policy. While this process is going on, measurement tools such as Pingparser are used to collect values, which are plugged into the formula to determine the capacity of each participating CSP, through this, such metrics as volume of processed data, IP transfer delays, packet loss frequency and so on are derived and these are regarded as CSPs behaviour. There are other metrics used in determining the behaviours of different CSPs in a larger table, but selected these as they relate more to the test under review. The DL Time is time taken to download a share of 1638 bytes from each CSP. It includes time taken for each CSP to receive an API call, authentication of user, access, through to the time taken for the algorithm to receive the share for symmetric key recovery.

This time varies for all the CSPs with AWS Canada being the fastest and AWS Mumbai taking longer time than others. A closer look provides the understanding that some metrics affect the arrival time such as the Throughput, Download bandwidth and Speed as latency are virtually same across bar with minor difference between them while Availability is 99.9% as none of them failed during the operations. The explanation to this differences is in the period of the experiment from 19:12hours through to 19:48 hours. Due to different time zones of the CSPs, the time varied from peak to off-peak periods depending on the time zone. Thus underscoring the importance of using more metrics to calculate cloud’s level of cooperation as such are subjective to other metrics.

#### 6.3.1.2 Test Two

This experiment tests the fairness of using the two capacities mentioned in test one to disenroll participant. It tends to provide evidence that supports an argument that using the two variables above to judge cloud behaviour is fair. Implying that using such without looking for future likely results are sufficient to determine level of cooperation of each CSP.

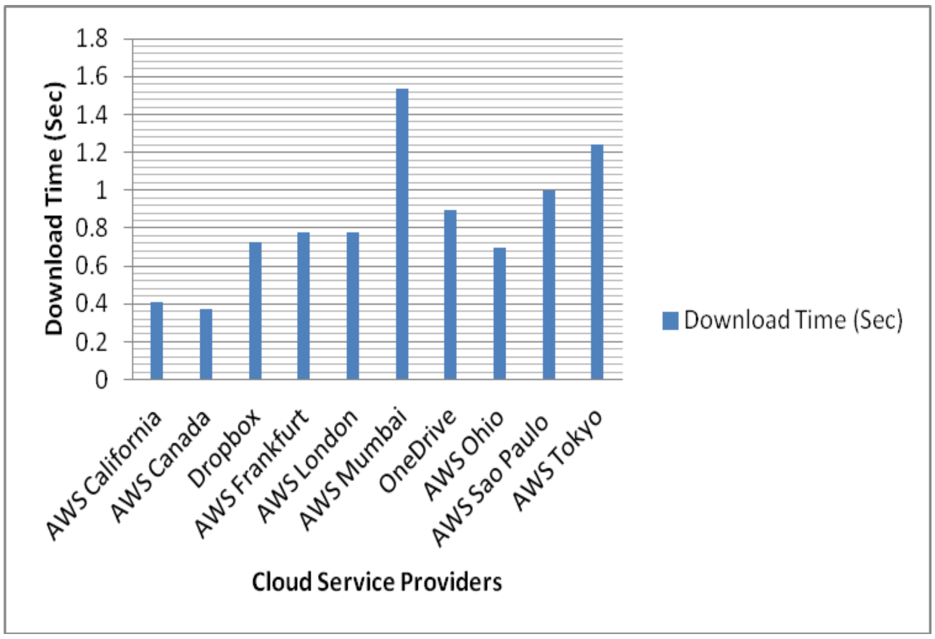


**Figure 17: Plot of CSPs against Download Time in seconds.**

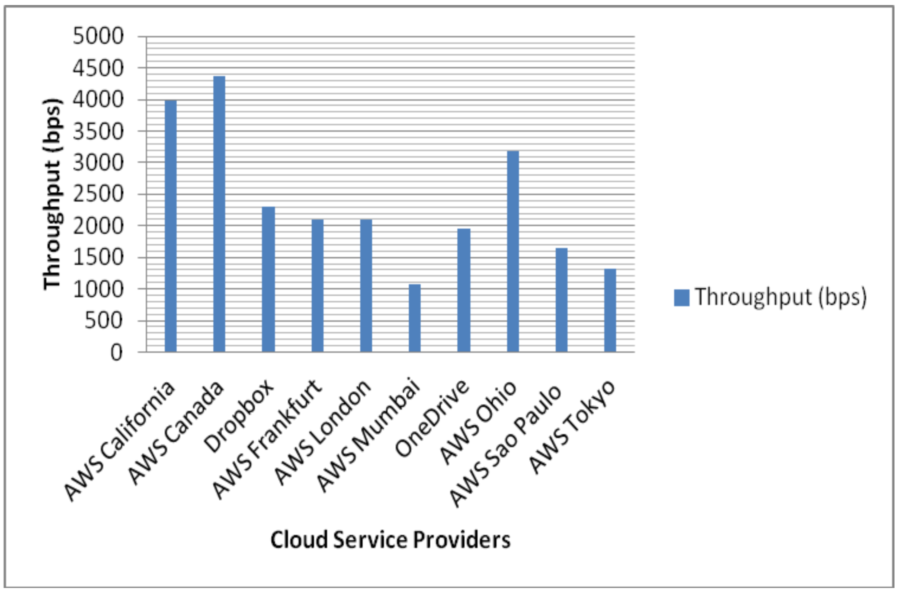


**Figure 18: Measuring the Throughputs of different CSPs**

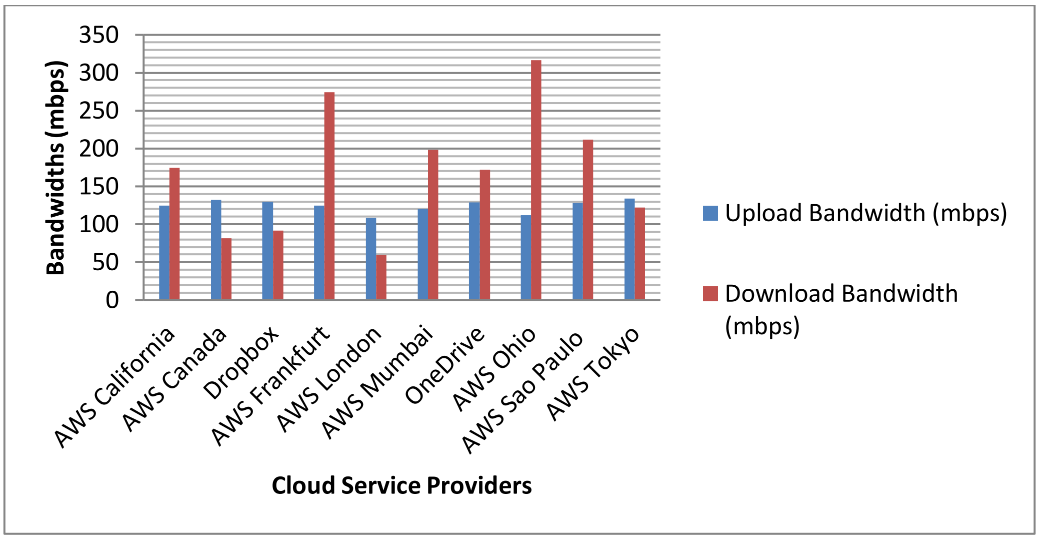


**Figure 19: Upload & Download Bandwidths of different CSPs**

The results in Figures 17, 18, and 19 show the fluctuating nature of download time (arrival time) alongside that of throughput. This is as a result of different time zones and the multi-tenant nature of CSPs and hence portends that values are unstable and the need for cumulative results both present and future in determining cloud behaviours. Take for instance, in Figure 18, Throughput of Canada is higher than that of London, which is closer to Edinburgh – the location of the experiment because of different time zones with regards to the period of the experiment. Therefore, concludes that using such attributes without looking into future results is not fair in application.

### 6.3.1.3 Test Three

This test also assumes that cloud outage will increase key recovery overheads and an increase in cloud outage leads to higher overheads.

**Table 11: Cloud Outages and Normal Situations**

| FragSizKeyShaPol | Key Recovery (S) | % Difference (S) |
|---|---|---|
| 15B, 3 from 5 | 7.80 | 16.41% Faster 20% failure |
| 15B, 3 from 5, 1 down | 6.52 | |
| 1.5KB, 3 from 5 | 7.76 | 51.80% Faster 40% failure |
| 1.5KB, 3 from 5, 2 down | 3.74 | |
| 15B, 6 from 10 | 25.67 | 37.90% Faster 30% failure |
| 15B, 6 from 10, 3 down | 15.94 | |
| 1.5KB, 6 from 10 | 25.33 | 42.99% Faster 40% failure |
| 1.5KB, 6 from 10, 4 down | 14.44 | |

The results in Table 11 above show that cloud outage has no negative effect on key recovery, rather reduces the overhead in comparison with normal situations. It shows the relationship between cloud outage and normal operational conditions. From available results at twenty percent (20%) failure rate using 3 from 5 share policy, the system becomes faster by sixteen percent (16.41%), but at forty percent (40%) failure rate using same share policy, the download speed is faster by close to fifty two percent (51.80%). Looking at a higher share policy of 6 from 10, at thirty percent (30%) failure rate, the system download speed is higher by a little above thirty-seven percent (37.90%), while at forty percent (40%) failure rate, the system performed better by about forty-three percent (42.99%). The implications therefore are that in as much as failure rate is not equivalent or above the threshold, system performance improves as there was no result obtained when the cloud outage exceeds the defined threshold. These therefore do not support the assumption as above that cloud outage has negative effect in key recovery. There is no significant evidence to show that the size of the share has effect on the key recovery during cloud outages because at forty percent (40%) failure rate using share of 10KB in 3 from 5 shows performance rate of above fifty-one percent while in 6 from 10 share policy approximately forty-three (42.99%) percent performance rate.

## 6.4 Overall Evaluation

Managing encryption keys is a very complex system [28], [131], and this complexity comes from the methods of generation, exchange, storage, retrieval, and replacement of cryptographic keys [131]. Hu *et al.* [132] believes that such complexity is more on retrieval process as the more complex the retrieval interface is, the greater the challenge of deploying key retrieval in applications such as decrypting encrypted file. FSSS key management is a keyless system built on secret sharing scheme using multi-cloud architecture. There have been other similar methods such as that of that of Shor *et al.* [4], Kapusta *et al.* [18], [123], Li *et al.* [125], Hu *et al.* [78] and Loruenser *et al.* [25]. This evaluation will be based on developed evaluation framework on Section 4.4.2 and on how these methods validate or disagree with the works of Abdallah and Salleh [8, 134] on key share retrieval when $n < m$, their available information on the application on this concept in cloud-based key management. Finally, on how each of these methods, proposed to handle the adverse effects of cloud failures on key share retrievals so as to ensure consistent key retrieval.

Hu *et al.* [78] used secret sharing scheme in key management but performed multiple heterogeneous-storage system. Their use of secret sharing is only based on share policy of *2-from-3,* and the *3* shares were stored in local hard disk drive, cloud storage bucket and a trusted third party. Of the *3* shares normally dispersed, that of the trusted third party is hardly used, and they believed that loss of share in secret sharing scheme is a rare scenario. This view perhaps limited their scope of experiments and hence never conducted experiments to understand the behaviour of share retrieval during adverse situation such as storage loss or outage.

Shor *et al.* [4] believes that the only way to secure sensitive data before storing it in the cloud is to encrypt before storage. With an evaluation of the inherent trade-offs of securing data in remote storage as well as an end-to-end analysis of the current methods of securing data using secret sharing scheme and encryption-based schemes on a local cluster storage device and multi-cloud environments. Their results suggest that the bottlenecks in securing data has moved from that of computational overheads of encoding and random data generation to network, storage and availability as a result of hardware accelerated encryption methods and hence concluded that data encryption and management of keys with an efficient secret sharing scheme is optimal for multi-cloud environments.  In all, their concentration is on efficiency, using throughput and not on the resilience nature of such methods. This, therefore, limited their scope of research and hence not on adverse effects of using secret sharing scheme during excessive cloud outages.

Loruenser *et al.* [25] developed a system known as ARCHISTAR that applies multiple secret sharing schemes on data before dispersal. By using computational secret sharing scheme as proposed by Krawczyk [11], it applies symmetric encryption on data to be shared, then shares encryption key using Shamir's PSS [2] and with Rabin's IDA [40] disperses data to multiple storages. Unlike Hu *et al.* in [78], the performance of Shamir's PSS when used to share ChaCha20-Poly1305 encryption key using *3-from-4* and *3-from-7* share policies were shown but failed to further explore experimentally the behaviours during adverse condition such as when cloud failures exceeds the defined threshold.

Li *et al.* [125] uses Ramp Secret Sharing Scheme (RSSS) to manage convergent keys known as Dekey, the work of Loruenser *et al.* [25] is a bit similar to that of Kapusta *et al* [18], [123] in using secret sharing to secure non-convergent encryption keys. While Li *et al.* [125] posit that Dekey uses RSSS to provide a *tunable* key management to balance issues of confidentiality, reliability, storage overhead and performance, Kapusta *et al.* deployed a system that situates between Rabin's IDA [40] and Krawczyk CSSS [11]. Curiously there is no known information on the resilient nature of the key shares retrieval and the accrued overhead for each sharing policy combinations during different rates of cloud failure. Secondly, there is also no known proposal on how to improve the resilience nature of secret sharing when failure rate of *n* exceeds that of threshold *m*. These knowledge gaps are what FSSS key management provide. Through extensive experiments, it validated the works of Abdallah and Salleh in [8, 134]. The experiments also provided needed information on the behaviours of CSPs during each rate of cloud failure and with the future work based on an Agent proposes the feasible way of preventing adverse cloud failures from inhibiting key shares recovery using self-organisation protocol.

All the same, FSSS key management looks impracticable in managing large-scale data infrastructure as key shares are stored on different cloudlets scattered all over the world as this has the potential of adding more overheads to overall system performance as seen in the available results. These were only intended to help inform on possible overheads in doing so. In the same vein, from available results the use of same CSPs at different geographical locations can also add to performance overheads as distance can be a factor of performance lag on network bandwidth. So, a balance should be implemented in choosing CSPs for an optimum performance in key storage management.

## 6.5 Conclusions

Evaluations of FSSS key management system with other similar methods such as that of Kapusta *et al.* [18], [123], Li *et al.* [125], Hu *et al.* [78] and Loruenser *et al.* [25] showed that though they used secret sharing in key management at different levels, there is no other known information on performance overheads of using any secret sharing policy beside the work of Loruenser *et al.* [25], which showed the performance of Shamir's PSS when used to share ChaCha20-Poly1305 encryption key using *3-from-4* and *3-from-7* share policies. But it failed to explore further experimentally the behaviours during adverse condition such as when cloud failure exceeds the defined threshold, and these are insufficient to guide users in choosing best file size-share policy combination. Above that, no one proved the resilient nature of their key retrieval system and how they intend to improve on the inherent problem of impossible key retrieval with secret sharing schemes when defined threshold is exceeded by cloud outages. The above knowledge gaps were what FSSS key management system provided using this chapter.

# 7 Disaster Management Methods Evaluation

## 7.1 Introduction

There are many risks in moving data into public cloud environments along with an increasing threat around large-scale data and key leakages during cloud outages as encryption keys are stored alongside data in many cloud storage resources. This chapter aims to present some features of FSSS in comparison with similar methods that make it reliable, readily available and resilient, and how it tends to use these qualities to redefine disaster management using cloud-based resources from that of recovery from losses to mitigation against losses. The previous chapter has been used to prove how FSSS key management is resilient against cloud outages, this thesis will through this chapter show how this resilience improves on the system overall resilient nature and what we are doing to maintain consistent data availability as future works as well as possible application scenario of FSSS design methodology.

## 7.2 Cloud-based Disaster Recovery (DR) System

The cloud disaster recovery system is entirely different in approach to traditional disaster recovery system according to Klein [6]. Cloud-based system takes an integrated approach in which the virtual server gets bundled with the operating system, applications, software patches and data. Furthermore, the backing up and copying of the entire server to an off-site data centres through virtualisation take as little as minutes.

The virtual server does not depend on a particular hardware, which makes it easier to transfer safely and accurately from one data centre to another the operating system, applications, patches and data without reloading each component of the server. The advantages of Cloud-based DR over the traditional type range from reduced recovery time and complete data accuracy during data restoration and recovery. These are made possible because of its ability to implement full network replication, the synchronisation or mirroring of Virtual Machines (VM) at a remote site to ensure failover in the event of failure of the original site.

Cloud-based DR as in all cloud-based systems provides low-cost DR solutions because of the *pay-as-you-use* model. Cloud-based DR provides its subscribers both shared and dedicated DR services. Based on customers' choice, the benefits can accrue but in all, cloud-based DR offers low-cost services compared to traditional systems. It is this "resource-on-demand" and high degree of automation that made cloud-based DR very attractive. Despite these attractive economics, Wood *et al*, [99] argue that increased latency is a major barrier in using cloud data centres for DR, as other servers could have a large geographical separation from the primary site, which in no small measure could affect communication between them adversely. The limitation of data owners from having control over their data placement worsens this scenario, they argued.

In the same vein, Ji *et al.* [100] and [79] opined that the use of synchronous replication by Cloud-based DR does not in any way help every data write as the wide-area latency has negative impacts on its performance, forcing system administrators to consider asynchronous replication. This often trades-off loss of data for performance by replicating a consistent "snapshot" to the backup site. Asynchronous replication positively impacts performance as the primary site can come up even before the replication completes. However, this can lead to loss of disk writes at the primary site subsequent to the last replicated snapshot in case of a disaster. Current designs and implementations of cloud-based disaster management systems are focused on recovery after outages leading to some losses, leakages and shut downs depending on the level and length of outages. Hence, the concentration is on reducing the Recovery Time Objectives (RTO) of cloud-based disasters, which is the time it takes to recover from disaster after it has happened, ranging from microseconds, seconds, minutes, hours and sometimes days and weeks.

## 7.3 FSSS Resilience Method

FSSS is a method of multi-cloud storage system; the resilience nature will be evaluated based on data fragmentations, storage, retrieval and key management methods as well as the reliability. Available literature from these works, Loruenser *et al.* [25], Gu *et al.* [9], Joshi *et al.* [133], Bessani *et al.* [23] and Bowers *et al.* [85] all give credence that the use of redundancy technique improves reliability, availability and resilience of cloud-based storage. In the other hands, the use of secret sharing improves resilience of cloud-based storage in as much as the defined threshold is not exceeded by cloud outages, there are proves from these works, Abdallah and Salleh [19], [27], Ukwandu *et al.* [46], Buchanan *et al.* [31], Fabian *et al.* [12], Ermakova and Fabian [13] to buttress these points. FSSS combines both methods in data sharing and key management. In data sharing using Figures *9* and *13* showed how *3* different cloud storages (premised and external) store encrypted fragments, and as well used Figures *9* and *14* to show its key share storages, for which the number of cloud storages deployed are dependent on key sharing policy in use. This thesis will therefore use the following experiments to prove further the level or reliability, availability and resilience that FSSS provides.

## 7.3.1 Test Results and Evaluations

### 7.3.1.1 Test One

This test is carried out using varied share policies, varied file sizes, with equal number of fragments. This is to test the implications of varied share policies on system overhead. It assumes that increase in threshold and change in policy has negative effect on file combination.

There is no significant evidence to show that increasing number of participants in a share policy impacts negatively on file combination. Taking a look at the Table 12 and Figure 20, it is obvious that though they all had equal number of fragments, system overheads increased slightly though not consistent and significant with increasing number of participants from 5 to 10 participants across board and 1KB through 1GB file sizes.

**Table 12: Overhead Cost of File Combination using equal number of fragments with varied share policies**

| S/N | FileSize (B) | 15%, 2 from 5 (S) | 15%, 3 from 5 (S) | 15%, 4 from 5 (S) | 15%, 4 from 10 (S) | 15%, 6 from 10 (S) | 15%, 8 from 10 (S) |
|---|---|---|---|---|---|---|---|
| 1 | 1KB | 0.02 | 0.03 | 0.02 | 0.07 | 0.02 | 0.05 |
| 2 | 10KB | 0.08 | 0.03 | 0.02 | 0.04 | 0.02 | 0.12 |
| 3 | 100KB | 0.05 | 0.03 | 0.02 | 0.06 | 0.03 | 0.04 |
| 4 | 1MB | 0.04 | 0.04 | 0.03 | 0.06 | 0.04 | 0.11 |
| 5 | 10MB | 0.18 | 0.19 | 0.17 | 0.21 | 0.17 | 0.26 |
| 6 | 100MB | 1.87 | 1.59 | 1.74 | 1.76 | 1.50 | 1.78 |
| 7 | 1GB | 78.10 | 81.38 | 78.01 | 88.34 | 79.98 | 96.10 |

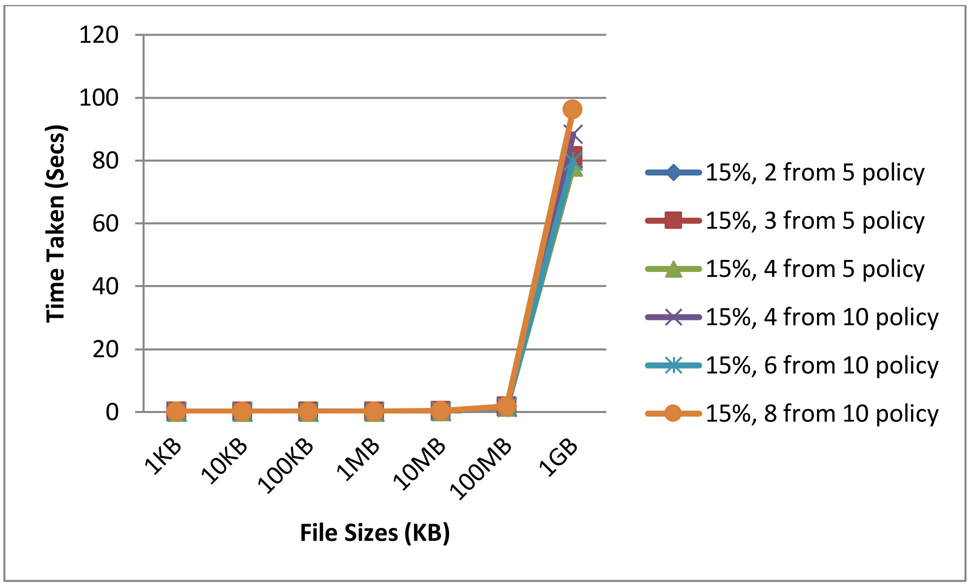


**Figure 20: Plot of Overhead Cost of File combination, equal number of fragments with varied share policies.**

In the same vein, an investigation to test the impact of varied thresholds, varied file sizes, equal number of fragments by seeking to know the relationship between thresholds and number of fragments generated keeping file sizes the same alongside number of participants. It is assumed here that the number of fragments generated has the capacity to cause large overheads in file combination. But a look at the above Table 12 and Figure 20 show that there is no significant evidence to show that increase in thresholds, increases the system overheads in an equal number of fragments using same file sizes and number of participants.

In order to make FSSS model more reliable and predictable, an equation was generated to help in this regard as shown below. This statistical model is designed to help users understand the time needed to process a given file in conjunction with the share policy. The essence of the model is to guide users in making an informed decision regarding the implication of choosing a share policy for a given file size.

#### 7.3.1.2 FSSS Model Equation

FSSS has two main inputs – file and share policy, while the system produces two categories of results that has to do with cost, that is the time in seconds it takes to process the file, which includes file split time using the File-splitter, fragment encryption time with AES 256-bit key length, fragment decryption time and file combination time using the File-combiner. The second category is the results in relation to key management using secret sharing policy, which accounts for key shares creation time using the sharing algorithm, key shares writing time, shares retrieval time, and secret recovery time using the share recovery algorithm.

Since the above experiments were performed using a finite blocks of file sizes (1KB, 10KB, 100KB, 1MB, 10MB, 100MB and 1GB), it is therefore necessary that a model that will help users understand the cost in seconds of what it will take to process their file given the file size as provided in Table 13. It is also worthy to note that the model is only limited to the share policies used in the experiments – 2 from 5, 3 from 5, 4 from 5, 4 from 10, 6 from 10 and 8 from 10.

**Table 13: File Processing Linear Regression table**

| Coefficients[a] | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Model | | Unstandardized Coefficients (S) | | Standardized Coefficients (S) | t (S) | Sig. (S) | Collinearity Statistics (S) | |
| | | B | Std. Error | Beta | | | Tolerance | VIF |
| 1 | (Constant) | -1.859 | 1.347 | | -1.381 | .176 | | |
| | File Size | 7.843E-8 | .000 | .993 | 53.801 | .000 | 1.000 | 1.000 |
| | P2 (3 from 5) | .420 | 1.872 | .005 | .224 | .824 | .600 | 1.667 |
| | P3 (4 from 5) | -.046 | 1.872 | -.001 | -.025 | .980 | .600 | 1.667 |
| | P4 (4 from 10) | 1.455 | 1.872 | .019 | .777 | .442 | .600 | 1.667 |
| | P5 (6 from 10) | .203 | 1.872 | .003 | .109 | .914 | .600 | 1.667 |
| | P6 (8 from 10) | 2.587 | 1.872 | .033 | 1.382 | .176 | .600 | 1.667 |
| a. Dependent Variable: OverHeadCost | | | | | | | | |

Equation modelling gives:

$$\text{OverHeadCost} = -1.859 + 0.00000007843 \times FS + 0.420 \times P2 - 0.046 \times P3 + 1.455 \times P4 + 0.203 \times P5 + 2.587 \times P6$$

FS is file size, P1 (reference variable) is share policy 2 from5, P2 is share policy 3 from 5, P3 is share policy 4 from 5, P4 is share policy 4 from 10, P5 is share policy 6 from 10, while P6 is share policy 8 from 10.

This equation model is derived from Table 13. The intrepretion is that P1 is the reference variable here, so it has been excluded from the equation. The interpretation has to be done in relation to P1.

For instance, given a file size, using share policy 3 from 5 in sharing and recovering a file within the limit of the system specification used in this experiment will take 0.42 seconds higher that when used P1 to share such a file, in the same vein, using share policy 4 from 5 to process same file size will be 0.046 seconds faster than when done with P1, while that of share policy 4 from 10 will take about 1.455 seconds higher than that of P1, 6 from 10 share policy will take about 0.203 seconds higher than that of P1 and 8 from 10 will take 2.587 seconds higher than that of P1. This implies in relation to our experimental frameworks (hardware and software specifications) that share policy 4 from 5 is more effective using any file of size and 15% of file size as fragment size, followed by 2 from 5, then 3 from 5, 6 from 10 and least effective is 8 from 10. When the overhead cost - that is the total time taken to process a file given a share policy is predicted, the other components such as file split time, fragment encryption time, fragment decryption time and file combination time predictions that made up the overhead cost will be generated as well. The linear regression models from which these are derived including that of share creation and secret recovery are added to this thesis as Appendixes.

#### 7.3.1.3 Test Two

This test assumes that an increase in the number of participants (cloudlets) increases file combination overhead and hence decreases the level of file availability. Table 12 shows these range of values: 1GB file size using share policy 2 from 5 took a total of 78.10 seconds to process, 3 from 5 took 81.38 seconds, while 4 from 5 took 78.01 seconds. A file of 1GB file size using share policy of 4 from 10 was processed at 88.34 seconds, 6 from 10 at 79.98 seconds while that of 8 from 10 took 96.10 seconds to process. This indicates that such assumption is false rather an increase in threshold increases system overhead. This implies that FSSS method has good level of availability as CSPs in use are scattered all over the world and are expected to have some impacts on share retrievals than when they are closely located.

#### 7.3.1.4 Test Three

This test the implication of cloud outages on file combination and hence assumes that cloud failure inhibits file combination and as such impossible to recombine file during cloud outage.

Table 11 in Test Three, Section 6.3.1.3 proved resilience of keys and in effect the data as also shown in Table 12. Evidence from Test Two, section 7.3.1 show that in as much as the threshold does not exceed 60% of total number of cloud storages in use, file reconstruction is always feasible, while evidence from Figure 11 gave credence to the use of redundancy approach, which improves data availability. Hence, it is pertinent to assert that the use of both methods in cloud storages of both encrypted fragments and key shares ensure consistent data availability, reliability and resilience.

## 7.4 Overall Evaluations

The evaluation on this chapter will be based on the potential application area of cloud-based disaster management. This thesis therefore will take a look at what is currently being done and evaluate them with the characteristics of FSSS.

Current cloud-based disaster management are based on recovery after cloud outages. Current works and proposal in cloud-based disaster management are based on minimising the time it takes to recover from disaster known as Recovery Time Objective (RTO), but the point of recovery known as Recovery Point Objective (RPO) has remained a management decision issue [79]. Issues of concern for evaluation are limited to, what are being done to automate the RPO decision point, mitigate losses rather than recovering after losses occasioned by disaster.

Gu *et al.* [9] relied on data backup and restore technology to build a system proposed to provide high data reliability, low backup cost and short recovery time by utilising co-operative resources of various cloud service providers with various parameters using multiple optimisation scheduling as strategies in balancing the objectives of disaster recovery. Lenk *et al.* [111] states that there is a way of being at alert against disasters by describing the architecture for a novel approach in establishing standby sites, known as warm sites in disaster recovery that replicates a distributed system in the cloud to another cloud they argued is a missing link in current literature. No doubt, their system is intended to provide a backup system in case of disaster.

Even with their deployment method, the certainties of downtime cannot be overemphasised. Cully *et al.* [67] described a system that provides software resilience in the face of hardware failure (VMs for Virtual Machines) in such a manner that an active system at such a time can continue execution on an alternative physical host while preserving the host configurations by using speculative execution in replicating either processor-intensive applications or communication-intensive applications at a fixed time interval. The strength lies on the preservation of system’s software independently during hardware failure but lags in the area of performing Replication by adding fault tolerance into the VM at fixed intervals.

Dong *et al.* [114] described an architecture known as Multi-cloud-based Evacuation Services for Emergency Management (MCES) that is based on instantiating multiple instances at different states as a way of mitigating the cloud-based disaster. Their work is similar to that of Chu and Wu [115], Chu and Wu [116] and Chen *et al.* [117].

Chu and Wu provided a hybrid system that combines cloud resources with mobile phones. The cloud serves for routine task computing while sensor information that provides the best evacuation routes for data are collected using mobile phones and these are done irrespective of an emergency situation or not. Chen *et al.* concentrated on mobile cloud computing using smart-phones. Thus, proving a point that MCES is designed for disaster recovery with its attendant downtimes depending on the time it takes to re-instantiate a new VM after the disaster.

By using the evaluation frameworks on resilience, it becomes obvious that when $Sp = 1$, the system breaking point has been reached and hence the needed information to begin recovery thereby achieving an automated RPO if self-organisation is not incorporated. In the same vein, using the proposed solution on how best to implement the concept of self-organisation as contained in future work, FSSS wishes to use same to bridge the gap on the RTO thereby achieving no downtime and as well prevent losses occasioned by outages. This, to the best of our knowledge is the first known attempt to shift the focus from disaster recovery to disaster mitigation. First, FSSS argues the sufficiency of using rate of availability and response time to determine cloud level of cooperation, as metrics like throughput, download bandwidth and speed affect the arrival time.

While availability is virtually 99.9% in modern cloud storage designs, the need to anticipate otherwise may not be needed as it has been covered in Service Level Agrement(SLAs) thus underscoring the importance of using more metrics to calculate cloud behaviours as such are subjective to other metrics. Secondly, its test on fairness of using the two capacities mentioned earlier to disenroll participant through Trust calculations. Using section 6.3.1 showed the fluctuating nature of download time (Arrival time) alongside that of throughput as a result of the multi-tenancy nature of CSPs and hence portends that values are unstable and the need for cumulative results both present and future in determining cloud behaviours. Therefore, concludes that using such attributes without looking into future results is not fair in application.

## 7.5 Conclusions

This chapter provides information on the tests conducted to present some features of FSSS in comparison with similar current methods like that of Gu *et al.* [9], Lenk *et al.* [111], Cully *et al.* [67], and Dong *et al.* [114]. While these methods aimed at reducing the impact of cloud outage by minimising the RTO and leaving RPO as management decision issue, FSSS we conclude is the first attempt known to us that provided a framework that has the capacity of automating RPO, it also focuses on redefining disaster management by introducing the concept of mitigation of losses rather than recovery from losses after outage. It proves to do these through qualities like high end reliability, availability, and proposed redefined resilience through self-organisation using an agent as contained in future work section.

# 8 Conclusions and Future Work

## 8.1 Thesis Summary

This thesis is an evaluation of a fragmented secret share architecture known as FSSS, which aims to provide an alternative that is capable of closing knowledge gap inherent in secret sharing schemes, such as the inability to continue production as file size increases. Fabian and Fabian [12], Ermakova and Fabian [13] and Alsolami and Boult in [17] all posit that secret sharing scheme is suitable for use in data sharing but failed to show that it is capable of continuing production when file sizes increased exponentially thus limiting its use in large-scale data infrastructure. By using this thesis' evaluation framework on scalability as defined above, the overall evaluation with other similar methods showed that FSSS was able to provide a more scalable alternative by combining data fragmentation using optimum fragment size with secret sharing scheme in key management.

Managing encryption keys is a very complex system [28], [131], and this complexity comes from the methods of generation, exchange, storage, retrieval, and replacement of cryptographic keys [131]. Hu *et al.* [132] believes that such complexity is more on retrieval process as the more complex the retrieval interface is, the greater the challenge of deploying key retrieval in applications such as decrypting encrypted file. But by using secret sharing scheme in key management, high level resilience is feasible and has the capacity of providing non-complex key management system. FSSS defined resilience in using secret sharing scheme in key management as the ability to overcome the facts that when $n < m$, key recovery become impossible. It developed an evaluation framework based on this and used same to evaluate the potentials of different methods in compliance with this and found out that though there have been other similar methods such as that of Kapusta *et al.* [18], [123], Li *et al.* [125], Hu *et al.* [78] and Loruenser *et al.* [25] only FSSS validated this experimentally in accordance with the works of Abdallah and Salleh in [8, 134] on key share retrieval when $n < m$. It also went further to show behaviours of system at different rates of cloud outages and proposes possible solution to this mentioned challenge using self-organising protocol as proposed by Nojoumian *et al*. in [34].

With these above features, FSSS when compared with the works of Gu *et al.* [9], Lenk *et al.* [111], Cully *et al* [67], and Dong *et al.* [114] showed that while these methods aimed at reducing the impact of cloud outage by minimising the recovery time objective (RTO) in cloud-based disaster management, FSSS has shown potential in redefining disaster management by introducing the concept of mitigation of losses rather than recovery from losses after outage. It proves to do these through its qualities like high end reliability, availability, and proposed redefined resilience through self-organisation using an Agent.

## 8.2 Main Findings

This thesis has the following as the primary findings:

- The inherent challenge in secret sharing scheme by its inability to continue production when file size increases exponentially can only be solved through alternative means such as combining secret sharing with other data fragmentation method.
- Using an optimum fragment size in data fragmentation gives an almost even overhead cost irrespective of the share policy in use as difference between each of the overheads at different file sizes are less than three seconds but with an understanding that using an optimum fragment size for all file sizes may increase overhead as file sizes increase hence the need for some reviews and future experiments in that direction.
- Redefining the concept of resilience in secret sharing scheme using self-organising protocol has the ability to provide service irrespective of a number of cloud outages, thus closing the knowledge gap that secret sharing cannot be used to provide services when cloud outages exceed defined threshold in cloud-based storage.
- Disaster management using cloud resources has been focused on recovery after cloud outages, and so resilience has been defined not on mitigating against the occasioned losses but on a quick recovery. This has been a source of worry for cloud users as such outages could lead to losses, leakages and sometimes complete closure of businesses, FSSS fills in that gap as it provides consistent data availability in the face of outages while maintaining high-level security, confidentiality, and integrity of stored data.

## 8.3 Limitations

FSSS is deficient in the area of fragments management as it does not implement a threshold scheme in encrypted fragments recovery nor does it have an inbuilt privacy protection and error correction mechanisms. This makes it difficult to detect as well as provide information on unauthorised access, correct corrupt encrypted fragment(s), provide details of stolen fragment(s), or lost one in transit. So, when there is a malicious intent on encrypted fragment(s) by way of corruption or stealing, fragment correction becomes impossible and file recombination fails integrity test using SHA-512 file checksum. The only possible way out is a rerun of the process using different fragment store. This impacts the system negatively by impeding the speed of the process.

Using secret sharing scheme has an inherent deficiency in such a manner that when $n < m$, secret recovery is impossible. This thesis could not implement the desired self-organisation protocol as proposed by Nojoumian *et al.* in [34] in key management that we believed is capable of ensuring that cloud failure rate does not exceed defined threshold.

FSSS models only predict expected time to process file given a policy and these policies are not dynamically chosen like files rather users have to choose from a select Share Policies, which makes users options in terms of Share Policies limited.

## 8.4 Future Work

Two main issues form this thesis future works: one is to apply erasure coding technique on encrypted file using optimum fragment size, which this thesis found more scalable than using block fragment size in use in erasure coding and two is to implement self-organising protocol on key management so as to provide consistent key retrieval that will ensure steady data availability. Below is current work done to help define the future work in the area of self-organisation as mechanism for applying erasure coding technique are already defined in several literature.

### 8.4.1 The Social Concept in Secret Sharing

The social concept in secret sharing as proposed by Nojoumian *et al.* [4, 5, 6] suggests that the resilient nature of secret sharing scheme can be strengthened by using it to develop a self-organising system as it concerns the use of cloud storage resources. Our proposed future woks will be presented here, which we intend to use to redefine the concept of socialisation in secret sharing as presented in Section 2.4.4. It is built around using an agent that computes cloud behaviours after every phase of secret sharing and recovery operations. Uses same to predict future behaviours of cloudlets and hence invoke the social tuning functions of taking away shares from non-cooperative to cooperative cloudlets.

### 8.4.2 Cloud Behavioural Computation

After every operation, the user is presented with two different computational options – capacity measures of each participating cloudlet or overall cloudlets performance during each operation. The essence is to give the user options of perusing behaviours of each cloudlet during the operation such as its Latency (ms), Speed (bps), Throughput (bps) and Availability. While in the other hand, the overall performance of all cloudlets in the areas of Elapsed Time (sec), Packet Loss, Average Round Trip Time (ms), Speed (bps), Download Bandwidth (bps), Upload Bandwidth (bps), Latency (ms), and Throughput (bps) for comparative analysis. We will therefore treat this section by looking at these measures in details as designed.

### 8.4.3 Capacity Measures of clouds

At every operation of FSSS, measures are taken to ascertain several behaviours of participating cloudlets so as to help set up benchmarks of capacities of participating cloudlets for every file size against share policy in relation to accrued performance overheads such as latency, rate of packet loss, transaction speed and throughput as optimum fragment size has been established through previous publication and unpublished experiments. With this one can say using a certain file size in conjunction with a fragment size and share policy, the system is capable of having certain overheads accrued. With these prediction of future behaviours as well as potential dangers can be done. With the proposed future works, self-organisation can be activated to avert any impending danger so as to keep the system consistently available. These behaviours are calculated through capacity measures as provided in Tables 2 and 3.

### 8.4.4 Proposed Architectural Design of the Self-Organising System

In Figure 21, shares are mapped into cloudlets and details of ShareIDs returned to metadata store. Hence, the system is configured and initialised. Figure 22 shows cloud performance measurements, analysis and possible system adjustments, while Figure 23 shows actions taken when results are interpreted to avert danger thereby creating a resilient and self-organising system capable of mitigating disaster.

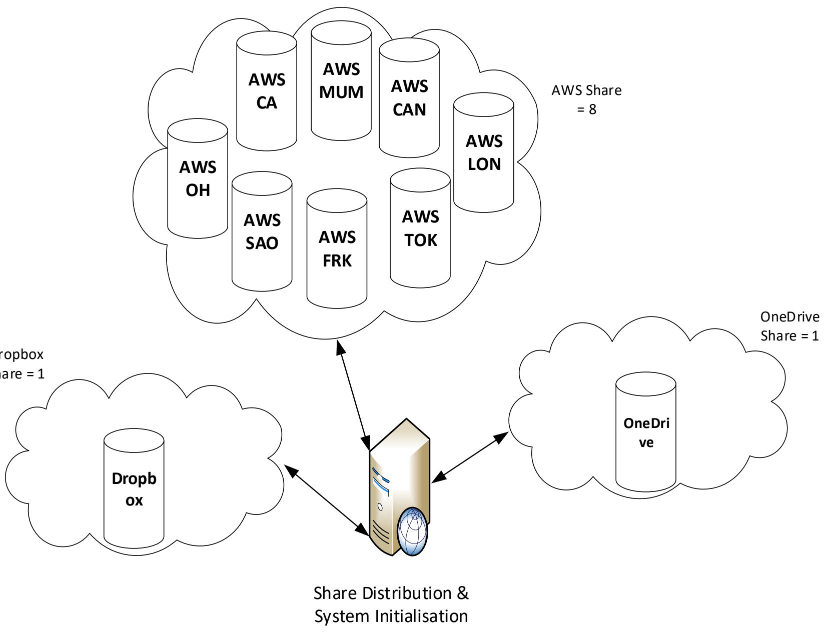


Figure 21: Share Mapping, Distribution and System Initialisation

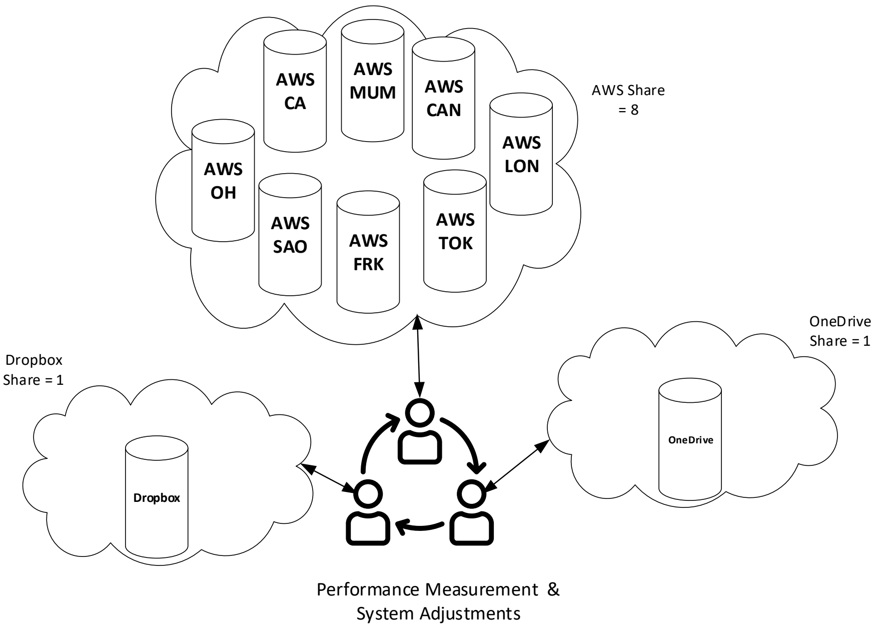


Figure 22: Performance Measurements and System Adjustments.

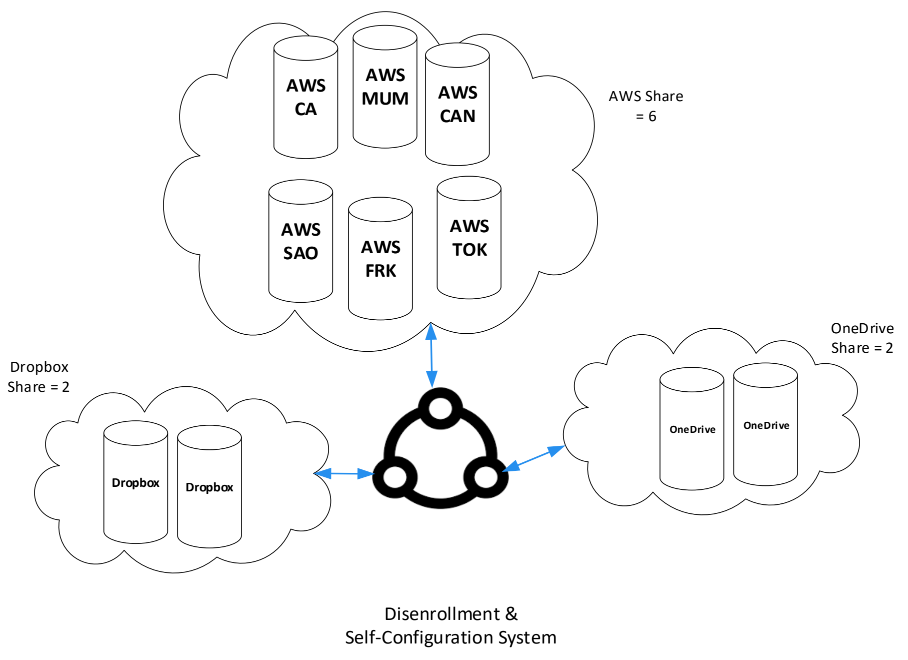


Figure 23: Disenrollment and Self-Reconfiguration System.

### 8.1.1 Agent

The activities of the proposed Agent is to receive data from the calculated cloudlets capacities from several storage files and use same based on the metrics to (1) Calculate cloud performance from measurements taken during each operations, analyse and possible issue pre-system adjustments notifications as shown in Figures 21 and 22. As shown in Figure 23, use values gathered to predict possible behaviours of cloudlets and where necessary actions are taken when results are interpreted as showing adverse behaviours by cloudlet. The actions are not limited to removing shares from such poorly behaved cloudlet and transferring same to better behaved cloudlet but not exceedingly pre-defined threshold. These adjustments thereby provide self-organisation that helps to prevent system failure due to adverse outages.

# 9 References

[1] H. Hassan, 'Organisational factors affecting cloud computing adoption in small and medium enterprises (SMEs) in service sector', *Procedia Comput. Sci.*, vol. 121, pp. 976–981, Jan. 2017.

[2] A. Shamir, 'How to share a secret', *Commun. ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[3] G. R. Blakely, 'Safeguarding cryptographic keys', in *Proc. AFIPS*, 1979, vol. 48, pp. 313–317.

[4] R. Shor, G. Yadgar, W. Huang, E. Yaakobi, and J. Bruck, 'How to Best Share a Big Secret', in *Proceedings of the 11th ACM International Systems and Storage Conference*, 2018, pp. 76–88.

[5] E. Ukwandu, W. J. Buchanan, and G. Russell, 'Performance evaluation of a fragmented secret share system', in *Cyber Situational Awareness, Data Analytics And Assessment (Cyber SA), 2017 International Conference On*, 2017, pp. 1–6.

[6] M. Klein, 'How the Cloud Changes Disaster Recovery, Industry Perspective', Jul. 2011.

[7] H. Kashiwazaki, 'Practical uses of cloud computing services in a Japanese university of the arts against aftermath of the 2011 Tohoku earthquake', in *Proceedings of the 40th annual ACM SIGUCCS conference on User services*, 2012, pp. 49–52.

[8] OnlineTech, 'Disaster Recovery White Paper.', 2013.

[9] Y. Gu, D. Wang, and C. Liu, 'DR-Cloud: Multi-cloud based disaster recovery service', *Tsinghua Sci. Technol.*, vol. 19, no. 1, pp. 13–23, Feb. 2014.

[10] M. Russ, 'Secret Sharing Schemes PowerPoint PPT Presentation'. 2012.

[11] H. Krawczyk, 'Secret sharing made short', in *Advances in Cryptology—CRYPTO'93*, 1993, pp. 136–146.

[12] B. Fabian, T. Ermakova, and P. Junghanns, 'Collaborative and secure sharing of healthcare data in multi-clouds', *Inf. Syst.*, vol. 48, pp. 132–150, 2015.

[13] T. Ermakova and B. Fabian, 'Secret sharing for health data in multi-provider clouds', in *Business Informatics (CBI), 2013 IEEE 15th Conference on*, 2013, pp. 93–100.

[14] Q. Zhang, S. Li, Z. Li, Y. Xing, Z. Yang, and Y. Dai, 'CHARM: A Cost-Efficient Multi-Cloud Data Hosting Scheme with High Availability', *IEEE Trans. Cloud Comput.*, vol. 3, no. 3, pp. 372–386, Jul. 2015.

[15] M. Thangapandiyan and P. M. R. Anand, 'Robust CHARM: an efficient data hosting scheme for cloud data storage system', *Autom. Control Comput. Sci.*, vol. 51, no. 4, pp. 240–247, Jul. 2017.

[16] T. Loruenser, A. Happe, and D. Slamanig, 'ARCHISTAR: towards secure and robust cloud based data sharing', in *Cloud Computing Technology and Science (CloudCom), 2015 IEEE 7th International Conference on*, 2015, pp. 371–378.

[17] F. Alsolami and T. E. Boult, 'CloudStash: using secret-sharing scheme to secure data, not keys, in multi-clouds', in *Information Technology: New Generations (ITNG), 2014 11th International Conference on*, 2014, pp. 315–320.

[18] K. Kapusta, G. Memmi, and H. Noura, 'An Efficient Keyless Fragmentation Algorithm for Data Protection', *ArXiv170509872 Cs*, May 2017.

[19] A. Abdallah and M. Salleh, 'Secret sharing scheme security and performance analysis', in *Computing, Control, Networking, Electronics and Embedded*

*Systems Engineering (ICCNEEE), 2015 International Conference on*, 2015, pp. 173–180.

[20] R. Koikara, D.-S. Kim, E.-J. Yoon, A. Paul, and K.-Y. Yoo, ‘Towards Security in Multi-clouds Using Secret Sharing’, *한국통신학회 학술대회논문집*, pp. 1557–1558, 2017.

[21] D. Pal, P. Khethavath, J. P. Thomas, and T. Chen, ‘Multilevel threshold secret sharing in distributed cloud’, in *International Symposium on Security in Computing and Communication*, 2015, pp. 13–23.

[22] H. Xiong, X. Zhang, D. Yao, X. Wu, and Y. Wen, ‘Towards End-to-end Secure Content Storage and Delivery with Public Cloud’, in *Proceedings of the Second ACM Conference on Data and Application Security and Privacy*, New York, NY, USA, 2012, pp. 257–266.

[23] A. Bessani, M. Correia, B. Quaresma, F. André, and P. Sousa, ‘DepSky: dependable and secure storage in a cloud-of-clouds’, *ACM Trans. Storage TOS*, vol. 9, no. 4, p. 12, 2013.

[24] F. Alsolami and T. E. Boult, ‘CloudStash: using secret-sharing scheme to secure data, not keys, in multi-clouds’, in *Information Technology: New Generations (ITNG), 2014 11th International Conference on*, 2014, pp. 315–320.

[25] T. Loruenser, A. Happe, and D. Slamanig, ‘ARCHISTAR: towards secure and robust cloud-based data sharing’, in *Cloud Computing Technology and Science (CloudCom), 2015 IEEE 7th International Conference on*, 2015, pp. 371–378.

[26] M. Li, C. Qin, J. Li, and P. P. C. Lee, ‘CDStore: Toward Reliable, Secure, and Cost-Efficient Cloud Storage via Convergent Dispersal’, *Internet Comput. IEEE*, vol. 20, no. 3, pp. 45–53, 2016.

[27] A. Abdallah and M. Salleh, ‘Analysis and comparison the security and performance of secret sharing schemes’, *Asian J. Inf. Technol.*, vol. 14, no. 2, pp. 74–83, 2015.

[28] B. Schneier, *Applied Cryptography: Protocols, algorithms, and source code in C*. John Wiley & Sons, 1996.

[29] C. E. Shannon and W. Weaver, ‘The mathematical theory of communication. 1949’, *Urbana Univ Ill. Press*, 1963.

[30] B. Schneier, *Applied cryptography: protocols, algorithms, and source code in C*. John Wiley & sons, 2007.

[31] W. J. Buchanan, D. Lanc, L. Fan, G. Russell, and others, ‘The Future Internet: A World of Secret Shares’, *Future Internet*, vol. 7, no. 4, pp. 445–464, 2015.

[32] M. Nojoumian and D. R. Stinson, ‘Brief announcement: secret sharing based on the social behaviors of players’, in *Proceedings of the 29th ACM SIGACT-SIGOPS symposium on principles of distributed computing*, 2010, pp. 239–240.

[33] M. Nojoumian, D. R. Stinson, and M. Grainger, ‘Unconditionally secure social secret sharing scheme’, *Inf. Secur. IET*, vol. 4, no. 4, pp. 202–211, 2010.

[34] M. Nojoumian and D. R. Stinson, ‘Social secret sharing in cloud computing using a new trust function’, in *Privacy, Security and Trust (PST), 2012 Tenth Annual International Conference on*, 2012, pp. 161–167.

[35] O. Farràs, T. Hansen, T. Kaced, and C. Padró, ‘On the Information Ratio of Non-Perfect Secret Sharing Schemes’, Cryptology ePrint Archive, Report 2014/124, 2014. https://eprint. iacr. org/2014/124. pdf, 2015.

[36] C. Asmuth and J. Bloom, ‘A modular approach to key safeguarding’, *IEEE Trans. Inf. Theory*, vol. 30, no. 2, pp. 208–210, 1983.

[37] E. F. Brickell, ‘Some ideal secret sharing schemes’, in *Advances in Cryptology—EUROCRYPT’89*, 1989, pp. 468–475.

[38] G. J. Simmons, ‘How to (really) share a secret’, in *Proceedings on Advances in cryptology*, 1990, pp. 390–448.

[39] B. Buchanan, *Cryptography*. River Publishers, 2017.

[40] M. O. Rabin, ‘Efficient dispersal of information for security, load balancing, and fault tolerance’, *J. ACM JACM*, vol. 36, no. 2, pp. 335–348, 1989.

[41] J. Benaloh and J. Leichter, ‘Generalized secret sharing and monotone functions’, in *Proceedings on Advances in cryptology*, 1990, pp. 27–35.

[42] P. Morillo, C. Padró, G. Sáez, and J. L. Villar, ‘Weighted threshold secret sharing schemes’, *Inf. Process. Lett.*, vol. 70, no. 5, pp. 211–216, 1999.

[43] A. Beimel, T. Tassa, and E. Weinreb, ‘Characterizing ideal weighted threshold secret sharing’, in *Theory of Cryptography*, Springer, 2005, pp. 600–619.

[44] S. Yoo, P. Park, J. Shin, and J. Ryou, ‘Key sharing scheme based on one weighted threshold secret sharing’, in *Advanced Communication Technology (ICACT), 2013 15th International Conference on*, 2013, pp. 317–320.

[45] A. Herzberg, S. Jarecki, H. Krawczyk, and M. Yung, ‘Proactive secret sharing or: How to cope with perpetual leakage’, in *Advances in Cryptology—CRYPT0’95*, Springer, 1995, pp. 339–352.

[46] E. Ukwandu, W. J. Buchanan, L. Fan, G. Russell, and O. Lo, ‘RESCUE: Resilient Secret Sharing Cloud-Based Architecture’, in *Trustcom/BigDataSE/ISPA, 2015 IEEE*, 2015, vol. 1, pp. 872–879.

[47] S. Takahashi and K. Iwamura, ‘Secret sharing scheme suitable for cloud computing’, in *Advanced Information Networking and Applications (AINA), 2013 IEEE 27th International Conference on*, 2013, pp. 530–537.

[48] N. Al Ebri, J. Baek, and C. Y. Yeun, ‘Study on Secret Sharing Schemes (SSS) and their applications’, in *Internet Technology and Secured Transactions (ICITST), 2011 International Conference for*, 2011, pp. 40–45.

[49] M. Nojoumian and T. C. Lethbridge, ‘A new approach for the trust calculation in social networks’, in *E-Business and Telecommunication Networks*, Springer, 2006, pp. 64–77.

[50] Sian-Jheng Lin and Wei-Ho Chung, ‘An Efficient (n, k) Information Dispersal Algorithm for High Code Rate System over Fermat Fields’, *Commun. Lett. IEEE*, vol. 16, no. 12, pp. 2036–2039, 2012.

[51] H. Lahkar and M. R, ‘Towards High Security and Fault Tolerant Dispersed Storage System with Optimized Information Dispersal Algorithm’, *Int. J. Adv. Res. Comput. Sci.*, vol. 5, no. 6, Jul. 2014.

[52] J. . Resch and J. . Plank, ‘AONT-RS: Blending security and performance in dispersed storage systems’, *Proc USENIX FAST ‘11*, 2011.

[53] K. Noyes, ‘IBM zeroes in on unstructured data with Cleversafe buy’, *CIO*, 05-Oct-2015. [Online]. Available: https://www.cio.com/article/2989292/ibm-zeroes-in-on-unstructured-data-with-cleversafe-buy.html. [Accessed: 12-Dec-2017].

[54] A. S. Eldin, A. Ghalwash, and H. Elshandidy, ‘Enhancing packet forwarding in Mobile Ad hoc Networks by exploiting the Information Dispersal Algorithm’, in *Communications, Computers and Applications, 2008. MIC-CCA 2008. Mosharaka International Conference on*, 2008, pp. 20–25.

[55] M. Conti, E. Gregori, and G. Maselli, ‘Reliable and efficient forwarding in ad hoc networks’, *Ad Hoc Netw.*, vol. 4, no. 3, pp. 398–415, May 2006.

[56] Z. Eslami, N. Pakniat, and M. Nojoumian, ‘Ideal social secret sharing using Birkhoff interpolation method’, *Secur. Commun. Netw.*, vol. 9, no. 18, pp. 4973–4982, Dec. 2016.

[57] G. Traverso, D. Demirel, S. M. Habib, and J. Buchmann, ‘AS 3: Adaptive social secret sharing for distributed storage systems’, in *Privacy, Security and Trust (PST), 2016 14th Annual Conference on*, 2016, pp. 528–535.

[58] P. Mell and T. Grance, ‘The NIST definition of cloud computing’, 2011.

[59] I. M. Abbadi, *Cloud management and security*. John Wiley & Sons, 2014.

[60] G. Kaefer, ‘Cloud Computing Architecture’, *Siemens Httpwww Scribd Comdoc56483000Cloud-Comput.-Archit.-Gerald-Kaefer*, 2010.

[61] M. Pranav, "Supporting naming of undiscovered application services is really a useful addition Reply, ‘New VMware Cloud Management Application Visibility Update’, *VMware Cloud Management*, 03-May-2012. [Online]. Available: https://blogs.vmware.com/management/2012/05/new-vmware-cloud-management-application-visibility-update.html. [Accessed: 17-Mar-2018].

[62] I. Morozan, ‘Multi-clouds database: A new model to provide security in cloud computing’, *Vrije Univ.*, 2015.

[63] M. A. AlZain, B. Soh, and E. Pardede, ‘MCDB: Using multi-clouds to ensure security in cloud computing’, in *Dependable, autonomic and secure computing (DASC), 2011 IEEE Ninth International Conference on*, 2011, pp. 784–791.

[64] A. Patel and K. Soni, ‘Three Major Security Issues in Single Cloud Environment’, *Int. J.*, vol. 4, no. 4, 2014.

[65] C. Padsala, R. Palav, P. Shah, and S. Sonawane, ‘Survey of Cloud Security Techniques’, *Int. J. Res. Appl. Sci. Eng. Technol.*, vol. 3, no. 3, pp. 47–50, 2015.

[66] D. Mounica and M. C. R. Rani, ‘Optimized Multi-Clouds using Shamir Shares’, *Int J Dev Comput Sci Technol*, vol. 1, pp. 83–87, 2013.

[67] J. V. Bharambe and R. K. Makhijani, ‘Secured data storage and retrieval in multiclouding using Shamir’s secret sharing algorithm’, 2013.

[68] A. Mallareddy, V. Bhargavi, and K. D. Rani, ‘A Single to Multi-Cloud Security based on Secret Sharing Algorithm’, *Int. J. Res.*, vol. 1, no. 7, pp. 910–915, 2014.

[69] M. Padmavathi, D. Sirisha, and R. A. Lakshman, ‘The Security of Cloud Computing System Enabled by Shamir’s Secret Sharing Algorithm’, *Int J Res Stud Sci Eng Technol*, vol. 1, pp. 103–109, 2014.

[70] T. Makkena and T. Rao, ‘A Shamir Secret Based Secure Data sharing between Data owners’, 2014.

[71] T. Takagi and K. Morozov, ‘MEXT Secret Sharing and Cloud Computing Workshop Overview’, 2011.

[72] J. L. Dautrich and C. V. Ravishankar, ‘Security limitations of using secret sharing for data outsourcing’, in *Data and Applications Security and Privacy XXVI*, Springer, 2012, pp. 145–160.

[73] M. A. Hadavi and R. Jalili, ‘Secure data outsourcing based on threshold secret sharing; towards a more practical solution’, in *VLDB 2010 PhD Workshop, Singapore*, 2010, pp. 54–59.

[74] D. Agrawal, A. El Abbadi, F. Emekci, A. Metwally, and S. Wang, ‘Secure data management service on cloud computing infrastructures’, in *New Frontiers in Information and Software as Services*, Springer, 2011, pp. 57–80.

[75] X. Tian, C. Sha, X. Wang, and A. Zhou, ‘Privacy preserving query processing on secret share-based data storage’, in *Database Systems for Advanced Applications*, 2011, pp. 108–122.

[76] M. Tompa and H. Woll, ‘How to share a secret with cheaters’, *J. Cryptol.*, vol. 1, no. 3, pp. 133–138, 1989.

[77] K. M. Martin, ‘Challenging the adversary model in secret sharing schemes’, *Coding Cryptogr. II Proc. R. Flem. Acad. Belg. Sci. Arts*, pp. 45–63, 2008.

[78] L. Hu, Y. Huang, D. Yang, Y. Zhang, and H. Liu, ‘SSeCloud: Using secret sharing scheme to secure keys’, in *IOP Conference Series: Earth and Environmental Science*, 2017, vol. 81, p. 012207.

[79] M. A. Khoshkholghi, A. Abdullah, R. Latip, S. Subramaniam, and M. Othman, ‘Disaster recovery in cloud computing: A survey’, *Comput. Inf. Sci.*, vol. 7, no. 4, p. 39, 2014.

[80] S. Rajagopalan, B. Cully, R. O'Connor, and A. Warfield, 'SecondSite: disaster tolerance as a service', in *ACM SIGPLAN Notices*, 2012, vol. 47, pp. 97–108.

[81] B. Cully, G. Lefebvre, D. Meyer, M. Feeley, N. Hutchinson, and A. Warfield, 'Remus: High availability via asynchronous virtual machine replication', in *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, 2008, pp. 161–174.

[82] M. C. Caraman, S. A. Moraru, S. Dan, and D. M. Kristaly, 'Romulus: disaster tolerant system based on Kernel Virtual Machines', *Ann. DAAAM Proc.*, pp. 1671–1673, 2009.

[83] A. Juels and B. S. Kaliski Jr, 'PORs: Proofs of retrievability for large files', in *Proceedings of the 14th ACM conference on Computer and communications security*, 2007, pp. 584–597.

[84] G. Ateniese *et al.*, 'Provable data possession at untrusted stores', in *Proceedings of the 14th ACM conference on Computer and communications security*, 2007, pp. 598–609.

[85] K. D. Bowers, A. Juels, and A. Oprea, 'HAIL: a high-availability and integrity layer for cloud storage', in *Proceedings of the 16th ACM conference on Computer and communications security*, 2009, pp. 187–198.

[86] Q. Wang, C. Wang, K. Ren, W. Lou, and J. Li, 'Enabling public auditability and data dynamics for storage security in cloud computing', *Parallel Distrib. Syst. IEEE Trans. On*, vol. 22, no. 5, pp. 847–859, 2011.

[87] L. Wei *et al.*, 'Security and privacy for storage and computation in cloud computing', *Inf. Sci.*, vol. 258, pp. 371–386, 2014.

[88] Y. Tang, P. P. Lee, J. C. Lui, and R. Perlman, 'FADE: Secure overlay cloud storage with file assured deletion', in *Security and Privacy in Communication Networks*, Springer, 2010, pp. 380–397.

[89] Y. Tang, P. P. Lee, J. Lui, and R. Perlman, 'Secure overlay cloud storage with access control and assured deletion', *Dependable Secure Comput. IEEE Trans. On*, vol. 9, no. 6, pp. 903–916, 2012.

[90] B. Calder *et al.*, 'Windows Azure Storage: a highly available cloud storage service with strong consistency', in *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, 2011, pp. 143–157.

[91] S. Guilloteau and M. Venkatesen, *Privacy in Cloud Computing-ITU-T Technology Watch Report March 2012*. 2013.

[92] G. Kulkarni, R. Waghmare, R. Palwe, V. Waykule, H. Bankar, and K. Koli, 'Cloud storage architecture', in *Telecommunication Systems, Services, and Applications (TSSA), 2012 7th International Conference on*, 2012, pp. 76–81.

[93] S. Subashini and V. Kavitha, 'A survey on security issues in service delivery models of cloud computing', *J. Netw. Comput. Appl.*, vol. 34, no. 1, pp. 1–11, 2011.

[94] Y. Wang, Z. Li, and Y. Sun, 'Cloud computing key management mechanism for cloud storage', 2015.

[95] B. T. Rao, 'A study on data storage security issues in cloud computing', *Procedia Comput. Sci.*, vol. 92, pp. 128–135, 2016.

[96] R. V. Rao and K. Selvamani, 'Data Security Challenges and Its Solutions in Cloud Computing', *Procedia Comput. Sci.*, vol. 48, pp. 204–209, Jan. 2015.

[97] D. Zissis and D. Lekkas, 'Addressing cloud computing security issues', *Future Gener. Comput. Syst.*, vol. 28, no. 3, pp. 583–592, 2012.

[98] P. Gharjale and P. Mohod, 'Efficient public key cryptosystem for scalable data sharing in Cloud storage', in *Computation of Power, Energy Information and Commuincation (ICCPEIC), 2015 International Conference on*, 2015, pp. 0325–0329.

[99] T. Wood, E. Cecchet, K. K. Ramakrishnan, P. J. Shenoy, J. E. van der Merwe, and A. Venkataramani, 'Disaster Recovery as a Cloud Service: Economic Benefits & Deployment Challenges.', *HotCloud*, vol. 10, pp. 8–15, 2010.
[100] M. Ji, A. C. Veitch, and J. Wilkes, 'Seneca: remote mirroring done write.', in *USENIX Annual Technical Conference, General Track*, 2003, pp. 253–268.
[101] V. Javaraiah, 'Backup for cloud and disaster recovery for consumers and SMBs', in *Advanced Networks and Telecommunication Systems (ANTS), 2011 IEEE 5th International Conference on*, 2011, pp. 1–3.
[102] M. Pokharel, S. Lee, and J. S. Park, 'Disaster recovery for system architecture using cloud computing', in *Applications and the Internet (SAINT), 2010 10th IEEE/IPSJ International Symposium on*, 2010, pp. 304–307.
[103] J. I. Khan and O. Y. Tahboub, 'Peer-to-Peer Enterprise Data Backup over a Ren Cloud', in *Information Technology: New Generations (ITNG), 2011 Eighth International Conference on*, 2011, pp. 959–964.
[104] Z. Jian-Hua and Z. Nan, 'Cloud computing-based data storage and disaster recovery', in *Future Computer Science and Education (ICFCSE), 2011 International Conference on*, 2011, pp. 629–632.
[105] S. R. Patil, R. M. Shiraguppi, B. P. Jain, and S. Eda, 'Methodology for Usage of Emerging Disk to Ameliorate Hybrid Storage Clouds', in *IEEE International Conference on Cloud Computing in Emerging Markets (CCEM)*, 2012, pp. 1–5.
[106] Y. Ueno, N. Miyaho, S. Suzuki, and K. Ichihara, 'Performance Evaluation of a Disaster Recovery System and Practical Network System Applications', in *Systems and Networks Communications (ICSNC), 2010 Fifth International Conference on*, 2010, pp. 195–200.
[107] T. Wood, H. A. Lagar-Cavilla, K. K. Ramakrishnan, P. Shenoy, and J. Van der Merwe, 'PipeCloud: using causality to overcome speed-of-light delays in cloud-based disaster recovery', in *Proceedings of the 2nd ACM Symposium on Cloud Computing*, 2011, p. 17.
[108] Y. Nakajima *et al.*, 'Design and implementation of virtualized ICT resource management system for carrier network services toward cloud computing era', in *ITU Kaleidoscope: Building Sustainable Communities (K-2013), 2013 Proceedings of*, 2013, pp. 1–8.
[109] N. Aghdaie and Y. Tamir, 'Fast transparent failover for reliable web service', in *Proceedings of the 15th IASTED international conference on parallel and distributed computing and systems (PDCS)*, 2003, pp. 757–762.
[110] M. Vrable, S. Savage, and G. M. Voelker, 'Cumulus: Filesystem backup to the cloud', *ACM Trans. Storage TOS*, vol. 5, no. 4, p. 14, 2009.
[111] A. Lenk and S. Tai, 'Cloud standby: disaster recovery of distributed systems in the cloud', in *European Conference on Service-Oriented and Cloud Computing*, 2014, pp. 32–46.
[112] A. Lenk, 'Cloud Standby Deployment: A Model-Driven Deployment Method for Disaster Recovery in the Cloud', in *Cloud Computing (CLOUD), 2015 IEEE 8th International Conference on*, 2015, pp. 933–940.
[113] M. P. da Silva, R. R. Obelheiro, and G. P. Koslovski, 'Adaptive Remus: adaptive checkpointing for Xen-based virtual machine replication', *Int. J. Parallel Emergent Distrib. Syst.*, vol. 32, no. 4, pp. 348–367, Jul. 2017.
[114] M. Dong, H. Li, K. Ota, L. T. Yang, and H. Zhu, 'Multicloud-Based Evacuation Services for Emergency Management', *IEEE Cloud Comput.*, vol. 1, no. 4, pp. 50–59, Nov. 2014.
[115] L. Chu and S. J. Wu, 'An Integrated Building Fire Evacuation System with RFID and Cloud Computing', in *2011 Seventh International Conference on*

*Intelligent Information Hiding and Multimedia Signal Processing*, 2011, pp. 17–20.

[116] L. Chu and S.-J. Wu, ‘A real-time decision support with cloud computing based fire evacuation system’, in *The 16th North-East Asia Symposium on Nano, Information Technology and Reliability*, 2011, pp. 45–48.

[117] Y.-J. Chen, C.-Y. Lin, and L.-C. Wang, ‘A personal emergency communication service for smartphones using FM transmitters’, in *Personal Indoor and Mobile Radio Communications (PIMRC), 2013 IEEE 24th International Symposium on*, 2013, pp. 3450–3455.

[118] P. Jogalekar and M. Woodside, ‘Evaluating the scalability of distributed systems’, *IEEE Trans. Parallel Distrib. Syst.*, vol. 11, no. 6, pp. 589–603, 2000.

[119] J. P. Sterbenz *et al.*, ‘Resilience and survivability in communication networks: Strategies, principles, and survey of disciplines’, *Comput. Netw.*, vol. 54, no. 8, pp. 1245–1265, 2010.

[120] Z. Li, L. O’brien, H. Zhang, and R. Cai, ‘On a catalogue of metrics for evaluating commercial cloud services’, in *Grid Computing (GRID), 2012 ACM/IEEE 13th International Conference on*, 2012, pp. 164–173.

[121] A. K. Bardsiri and S. M. Hashemi, ‘Qos metrics for cloud computing services evaluation’, *Int. J. Intell. Syst. Appl.*, vol. 6, no. 12, p. 27, 2014.

[122] S. Narani, ‘Social Secret Sharing for Resource Management in Cloud’, *ArXiv Prepr. ArXiv13021185*, 2013.

[123] K. Kapusta and G. Memmi, ‘A Fast Fragmentation Algorithm For Data Protection In a Multi-Cloud Environment’, *ArXiv Prepr. ArXiv180401886*, 2018.

[124] F. J. Alsolami, *Toward secure sensitive data in the cloud*. University of Colorado at Colorado Springs, 2015.

[125] J. Li, X. Chen, M. Li, J. Li, P. P. C. Lee, and W. Lou, ‘Secure Deduplication with Efficient and Reliable Convergent Key Management’, *IEEE Trans. Parallel Distrib. Syst.*, vol. 25, no. 6, pp. 1615–1625, Jun. 2014.

[126] R. Gracia-Tinedo, M. S. Artigas, A. Moreno-Martinez, C. Cotes, and P. G. Lopez, ‘Actively measuring personal cloud storage’, in *Cloud Computing (CLOUD), 2013 IEEE Sixth International Conference on*, 2013, pp. 301–308.

[127] E. Bocchi, I. Drago, and M. Mellia, ‘Personal cloud storage benchmarks and comparison’, *IEEE Trans. Cloud Comput.*, 2015.

[128] E. Bocchi, I. Drago, and M. Mellia, ‘Personal cloud storage: Usage, performance and impact of terminals’, in *Cloud Networking (CloudNet), 2015 IEEE 4th International Conference on*, 2015, pp. 106–111.

[129] P. Casas and R. Schatz, ‘Quality of experience in cloud services: survey and measurements’, *Comput. Netw.*, vol. 68, pp. 149–165, 2014.

[130] H. Jeon, Y.-G. Min, and K.-K. Seo, ‘A performance measurement framework of cloud storage services’, *Indian J. Sci. Technol.*, vol. 8, no. S8, pp. 105–111, 2015.

[131] F. Ogîgau-Neamtiu, "Cryptographic key management in cloud computing’, *Def. Resour. Manag. 21st Century*, 2015.

[132] X. Hu, J. An, N. Song, and S. Jingtao, ‘Key management scheme for bottleneck steiner tree based cloud computing’, 2015.

[133] G. Joshi, E. Soljanin, and G. Wornell, ‘Efficient Redundancy Techniques for Latency Reduction in Cloud Systems’, *ArXiv150803599 Cs Math*, Aug. 2015.

[134] A. Shamir, ‘How To Share a Secret’, *Commun. ACM CACM*, vol. 22, no. 1, pp. 612–613, 1979.

[135] W. Buchanan, D. Lanc, E. Ukwandu, L. Fan, and G. and, ‘The Future Internet: A World of Secret Shares’, *Future Internet*, vol. 7, no. 4, pp. 445–464, 2015.

[136] E. Ukwandu, W. J. Buchanan, and G. Russell, ‘TCloud: Availability at Zero Downtime’.

[137] A. Hoenes, D. McGrew, and P. Patnala, ‘Threshold Secret Sharing’. [Online]. Available: https://tools.ietf.org/html/draft-mcgrew-tss-03. [Accessed: 27-Jul-2018].

# 10 Appendix A

## 10.1 Performance Evaluation of a Fragmented Secret System

This work was done as an evaluation of the performance of RESCUE using desktop computers with folders as storage resources for the purpose of storing and recovering of shares. It serves as a way of testing the overall performance overhead so as to ascertain the implications of using RESCUE in data sharing in multi-cloud architecture. After these tests, RESCUE was scaled to full cloud implementation using Google Cloud VM alongside other cloud storage resources.

## 10.2 Overview of RESCUE

RESCUE is a secured threshold Cloud-based storage infrastructure using the Fragmented Secret Sharing System design philosophy, and is based on multi-cloud architecture for data storage. Replication for backup and restore of data from a primary site to other sites separated geographically shows little-known potential in eliminating system downtime because of the:

- **Effects of latency on performance:** the effect of latency on performance is a source of performance lag in using replication for backup and restore of data or virtualised infrastructure from a primary site to backup sites. Using synchronized replication in a multi-cloud storage system has an increasingly large overhead, and, on the other hand, asynchronous replication reduces the integrity of the replicated data.
- **Data integrity on recovery:** quality assurance of recovered data is an issue not readily discussed in data storage and retrieval, but a very strong necessity. So, checking the integrity of data after recovery is necessary to eliminate possible data corruption.
- **Consistent data availability:** data availability is key to the knowledge economy and therefore needful to mitigate factors that add large overheads to systems and thus using a robust, and all-encompassing, system is a necessity.

RESCUE is designed to handle: the latency effect on performance by defining the usage of key share mechanism rather than data sharing, when the file size is large.

It also addresses the issue of data integrity on recovery, as it is highly minimal or non-existent depending on the combination used in terms of file size, fragment size and key share policy. With regards to key share method; files are broken into chunks and the chunks in turn encrypted with AES and safely decrypted using recovered key before decrypting the chunks and combining file. In terms of shared data, data is treated as a sequence of bytes so data encoding does not matter and recovered file are cross-checked with the original file using SHA-512 hash function for data checksum. Additionally, using secret sharing scheme to split data and recover it assures data security in a keyless manner devoid of corruption as appropriate measure is put in place in the algorithm to detect wrong shares during data recovery.

## 10.3 Architecture

The architecture of RESCUE involves data/key splitting, storage, and retrievals. The two different methods involve data splitting or key splitting with data encryption and decryption using recovered key. The method implemented starts by defining the policy, which is the number of shares to be generated from each file, (N) and the least required number of shares (M) needed to come together to recover the file. The Policy of M-out-of-N, here the policy is 2-out-of-5, 3-out-of-5 and 4-out-of-5 shares, implying that for example 2 shares out-of-5 generated shares from a file are needed to recreate the file. The unique identifier is similar a magic number, unique to each session that is appended to the share when created.

### 10.3.1 File share

Files are scanned as in Figure 24 from a designated folder and encoded to byte streams. Using a pre-defined share policy, the encoded data is broken into shares. The shares generated are stored in separate containers and from where they are read-in and files recovered during file recovery on request.

### 10.3.2 Key share

Files are scanned as in Figure 24 from a designated folder as above, then using a predefined chunk size say 1024 Bytes, files are broken into the defined chunk size, encrypted and the encryption key shared as above. When the files are required, the shares generated from the key are brought together and the key recovered from where encrypted chunks are decrypted and the chunks brought together and the files are recreated.

#### 10.3.2.1 Share generation

Using the equation below we generate shares:

$$f(X, A) = \sum_{i=0}^{M-1} GF\_SUM \; A[i]X^{i} \qquad \textit{Equation 21}$$

To create N shares from a secret, with a threshold of M, we will take a look on how each octet of the secret is generated. An array A of M octets is created at first in which the array element A[0] contains a portion of the secret, while A[1], A[2],..., A[M-1] are selected independently and uniformly at random. Each share is generated by computing the value of *f*(*X*,A), where X is the share index and the resulting octet is appended to the share. A, B, C,... are arrays of *M* octets and each zero element of the array contains a portion of the share. A[0], B[0], C[0], .... are equal to first, second and third octets of the secret and so on. The power of X is the coefficient and M-1 is the threshold. GF_SUM is Galois field summation, which takes place over GF(256), different from integer addition as each addition uses the exclusive-or operation.

#### 10.3.2.2 Secret Recovery

Just as in Shamir [12] authorised participants following earlier stated rules are able to recover the secret using Lagrangian interpolation once the conditions:

1. All zero elements of the array of M octets are retrieved.
2. Number of retrieved elements greater or equal to the threshold.
3. All contributed shares from participants are certified as genuine and satisfies 2 above.

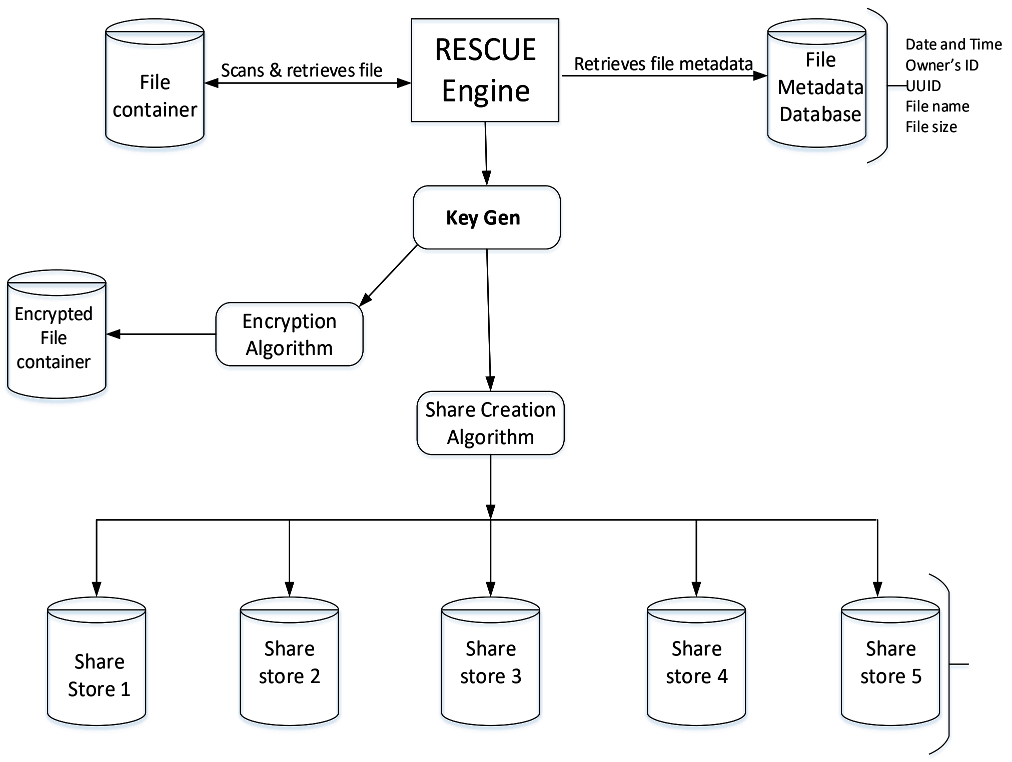


Figure 24: Key/File share creation

#### 10.3.2.3 Recovery: Files

When files are to be recovered as in Figure 25, the user types in the destination folder for recovered files; the program picks up each filename, the associated values that identify the owner of the files – the UUID all in the metadata database and used the values gathered to confirm ownership  and thereafter scans and retrieves all shares associated with the filename. With these, file recovery is made using the Recovery algorithm.

## 10.4 Recovery: Key

Following the initial method use above in ownership identification, the system retrieves the encrypted file, recover key and use the key so recovered to decrypt the file. See Figure 26 for details.

The number of shares recovered can be less than N but equals or greater than M (Threshold). The shares must be of equal length, else they are inconsistent. In file recovering, the output string is initialised to zero and the initial octet (share indexes are grouped in octets) of the share is stripped from each share and none of these octets are same else error will be reported, which halts the process. For each of these shares an array V of M octets is created, in which an array element V[i] contains the octet from ith share. These stripped octets are appended to the octets array U, formed by setting U[i] equals to the first octet of the *i*th share. The value of I(U, V) is computed, and appended to the output string, which is returned as the secret. This contains one fewer octet than the shares.

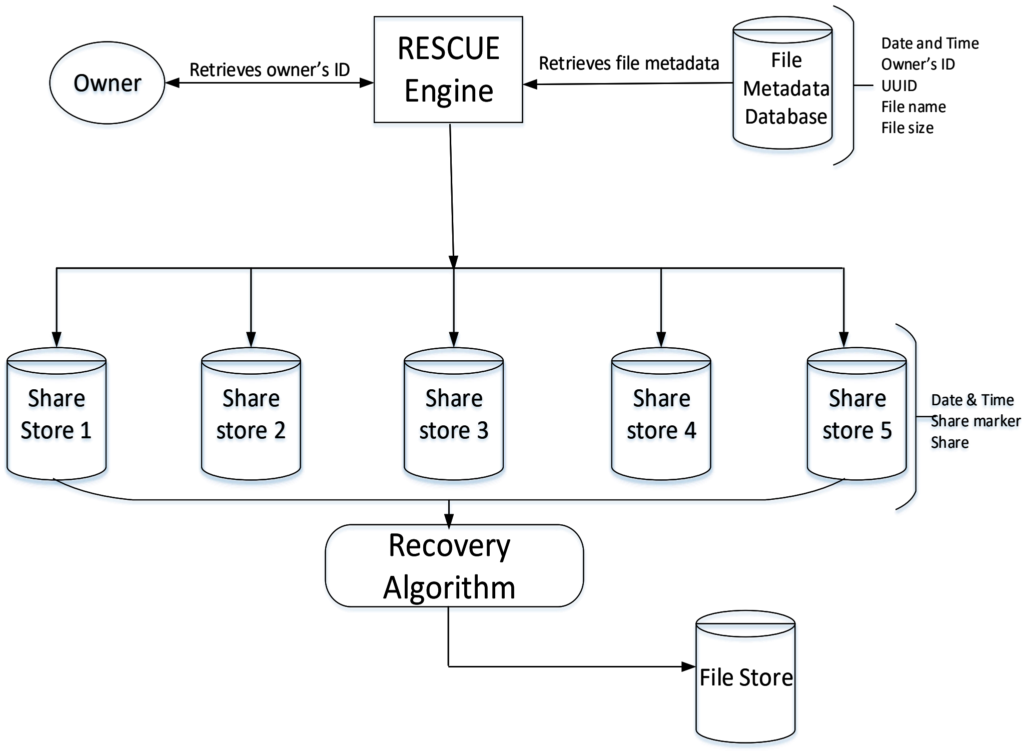


Figure 25: File Recovery

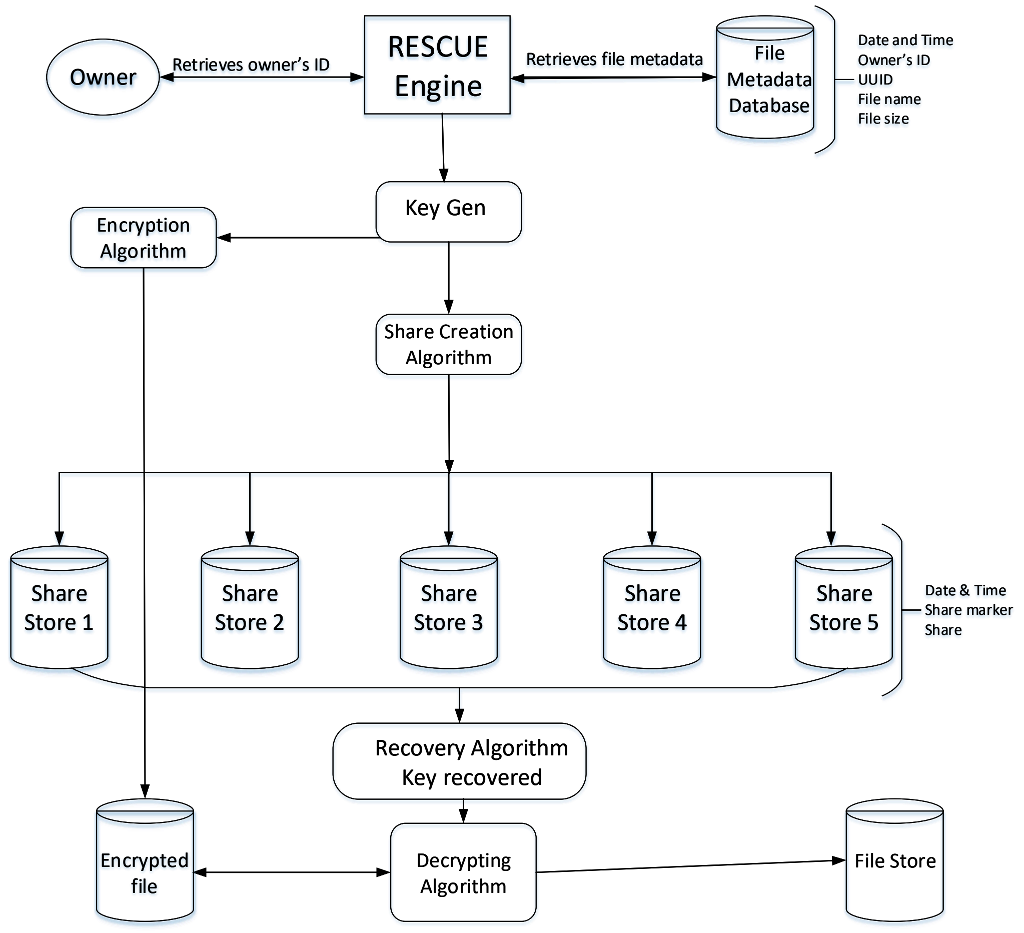


Figure 26: Key recovery and File decryption

## 10.5 Results and Evaluations

Two different sets of experiments were performed: file/data share; and key share methods. In file share, files of different sizes are created into share and stored in folders. When the files are needed, the several shares are recovered from the folders and the file recreated. Each file involved in the process is created into shares using M-out-of-N threshold secret sharing scheme and the shares stored in folders, while in key share, files of different sizes are broken into chunks; each chunk is encrypted using AES of 256-bits key length then stored in folder, the encryption key is thereafter shared, stored in folders as well.

When the files are needed, the shares are recovered from the folders for each key based on policy and the key recreated, using each key to decrypt a chunk as retrieved from the folder and the file recombined. The secret sharing scheme used is modified Social Secret sharing scheme. The issue of confidentiality and integrity in the use of secret sharing scheme has been validated by many works in secret sharing schemes such as Abdallah and Salleh [27], Buchanan *et al.* [135]. Since RESCUE is concentrated on Data Availability at Zero Downtime [136], the essence of the experiment is to understand all performance overheads that will impact negatively on the objective of the system so as to eliminate them or validating already known facts.

**Evaluation of the results:** Secret sharing schemes have been used successfully in data splitting and reconstruction, thereby providing data security in a keyless manner. This section outlines an experiment involving two main methods of secret sharing application – data sharing and key sharing. In Experiment One, figures 27 and 30 show normal curve with an increasing size of Threshold (M) and file size but Figure 28 and 29 showed otherwise, a varying curve indicating the effects of Share Writing and Recovery from folders on systems performance. In Experiment Two, Figures 31, 32 and 33 showed the validation of [13, 14] on the effect of increasing Threshold and file size on the system performance as in Figure 31, the threshold is 2, so the overhead is with the Process not on Recovery as in figures 32 and 33. But a look at Figure 34, 35 and 36 indicate entirely different results from the previous ones thus giving an understanding that there are resultant effects of file size, fragment size on share policy. The fragment size was varied in all as well as share policy using file sizes from 1KB to 1GB. In all the results shown, it is evident that using fragmented secret share system is the best option while dealing with big data infrastructure than using threshold secret sharing scheme alone, which has proved impossible to be used to scale large data infrastructure due to inherent characteristics of finite field arithmetic.

The evaluator, in this case, is the performance overhead at an increasing thresholds and data sizes. The experiments showed that Share Writing and Recovery adds more performance overhead in Experiments One, while in Experiment Two, the performance overheads of File and Fragment Sizes on Share Policy were obvious. These depict their strengths and weaknesses at different application scenarios.

The aim of the experiment is to discover all factors capable of adding performance overhead thereby derailing total system performance both in File and Key Sharing methods. Because we aim to apply the methods further in both network and cloud scenarios, we will work in eliminating the discovered factors that add to performance overhead to the system as this method has proved scalable with big data infrastructure. The test machine is a Duo Core Intel Pentium N3530 2.16GHz, 2.16 GHz, 64bit x64-based processor, Windows 8 operating system on 4GB of RAM.

Two primary sets of results are presented which use the parameters of *M=2, N=5; M=3, N=5 and M=4, N=5*. The variable *N* relates to the number of shares to create while the variable *M* relates to the number of shares required for recreation of the original arbitrary data (using each SSS algorithm). Results are presented in seconds for Time, while in KB, MB and GBs for variables file sizes. From the figures and tables presented, it can be clearly demonstrated that key share is the fastest method regardless of file sizes as well the method capable of scaling over large volumes of varying file sizes.

The key share experiment involves more stages than the previous and we therefore use the terms, **Process** and **Recover**. Process time involves time taken to split the file into chunks using a pre-defined chunk size, fragment encryption time, key share creation and writing times while Recover time involves time taken to recover key shares from folders, key recreation time, fragment decryption and file recombination times.

## 10.6 Conclusions

Experiments performed using secret sharing scheme has proved resilience in the face of failures as not all hosts are required to reconstruct data after splitting, but a major drawback remains the effect of latency on performance. This is worsened as data size increase as well as the distance between each of the hosts thus giving rise to our work. Lessons learnt are that using Key Share rather than Data Share method in combination with an appropriate fragment and share policy is the only way to scale large data infrastructure and with this lessons and validations we intend to eliminate all factors revealed as capable of adding large overhead to the system. This will provide a platform capable of achieving data availability at zero downtime.

## 10.7 Tables and Figures

Table 14: Share creation against policy

| | Policy: | 2 from 5 | 3 from 5 | 4 from 5 |
|---|---|---|---|---|
| S/N | File Size (KB) | Creation Time (Sec) | Creation Time (Sec) | Creation Time (Sec) |
| 1 | 1 | 0.106119 | 0.10933 | 0.143713 |
| 2 | 10 | 0.913352 | 1.075088 | 1.427096 |
| 3 | 100 | 1.833184 | 2.115918 | 2.461108 |

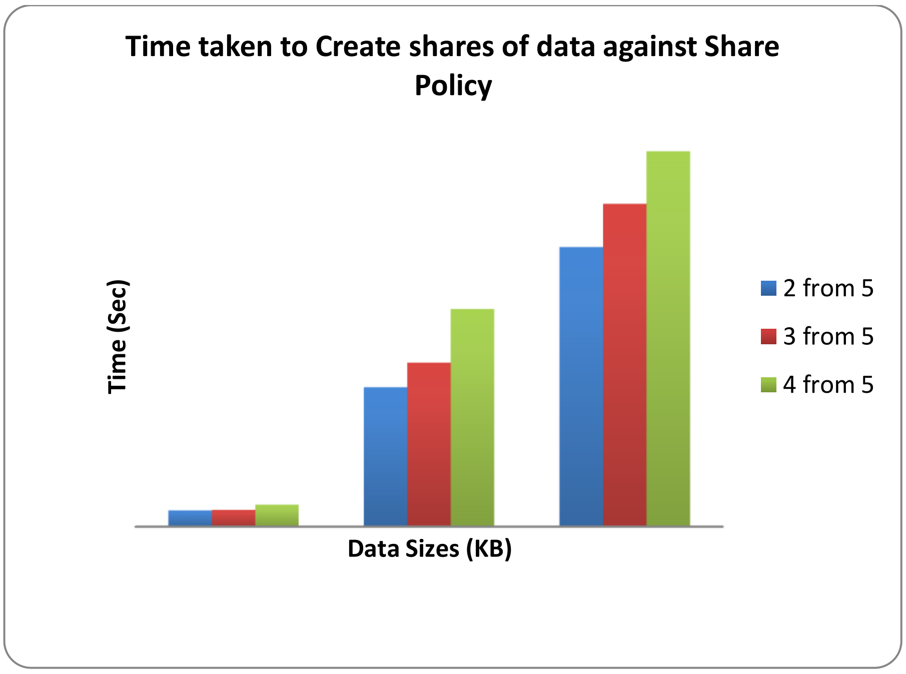


Figure 27: Time taken to Create share against Policy

Table 15: Share Writing to folders against Policy

|  | Policy: | 2 from 5 | 3 from 5 | 4 from 5 |
|---|---|---|---|---|
| S/N | File Size (KB) | Writing Shares (Sec) | Writing Shares(Sec) | Writing Shares(Sec) |
| 1 | 1 | 0.020532 | 0.03125 | 0.164257 |
| 2 | 10 | 0.066987 | 0.100468 | 0.03355 |
| 3 | 100 | 0.090945 | 0.085788 | 0.068099 |

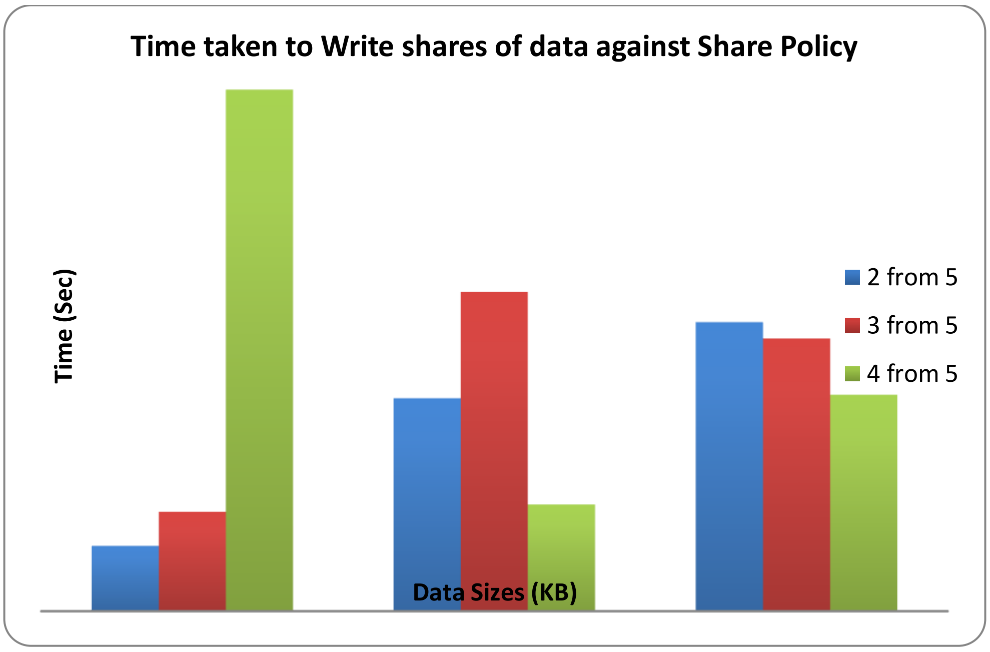


Figure 28: Share Writing to folders against Policy

Table 16:: Share Recovery against Policy

| | **Policy:** | **2 from 5** | **3 from 5** | **4 from 5** |
|---|---|---|---|---|
| S/N | File Size (KB) | Share Recovery (Sec) | Share Recovery (Sec) | Share Recovery (Sec) |
| 1 | 1 | 0.008113 | 0.004693 | 0.015012 |
| 2 | 10 | 0.083933 | 0.009362 | 0.005608 |
| 3 | 100 | 0.025136 | 0.010948 | 0.008912 |

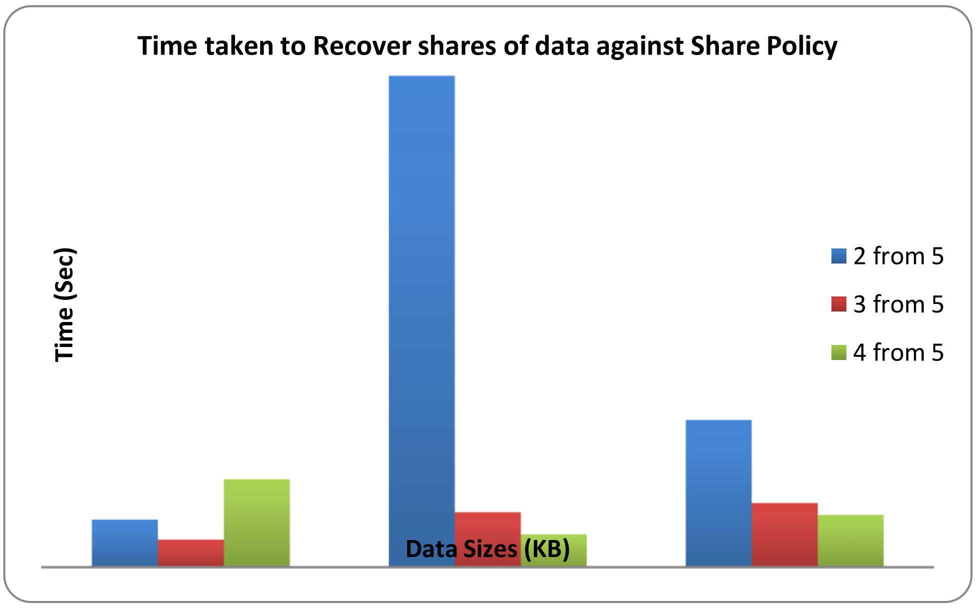


Figure 29: Share Recovery from folders against Policy

Table 17:: File Recreation against Policy

| | Policy: | 2 from 5 | 3 from 5 | 4 from 5 |
|---|---|---|---|---|
| S/N | File Size (KB) | File Recreation (sec) | File Recreation (sec) | File Recreation (sec) |
| 1 | 1 | 0.03405 | 0.054628 | 0.10265 |
| 2 | 10 | 0.434176 | 0.558682 | 0.92636 |
| 3 | 100 | 0.674842 | 1.091936 | 1.704002 |

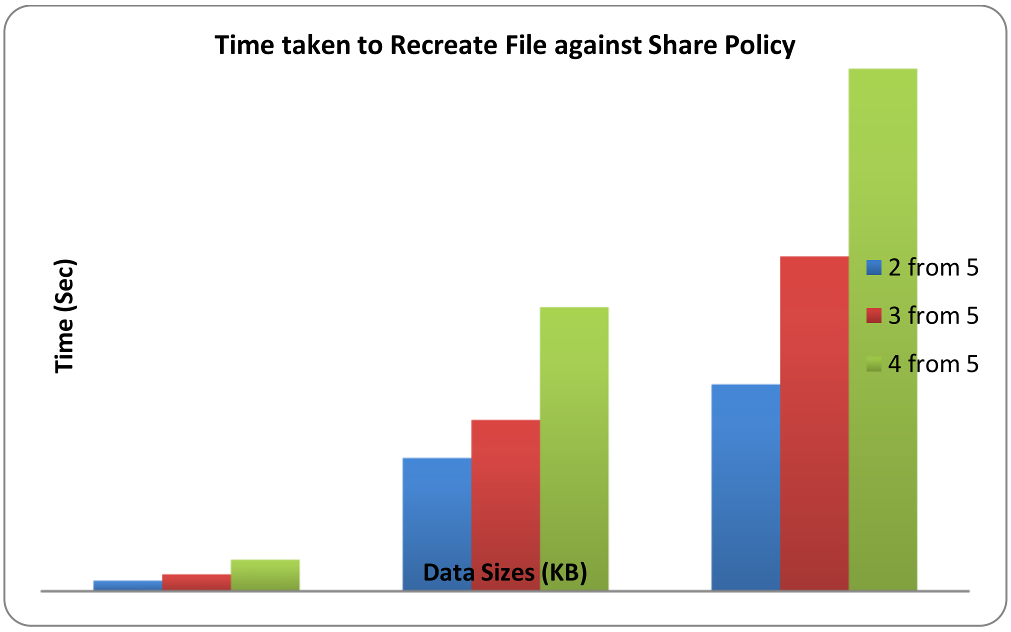


Figure 30: File Recreation against Policy

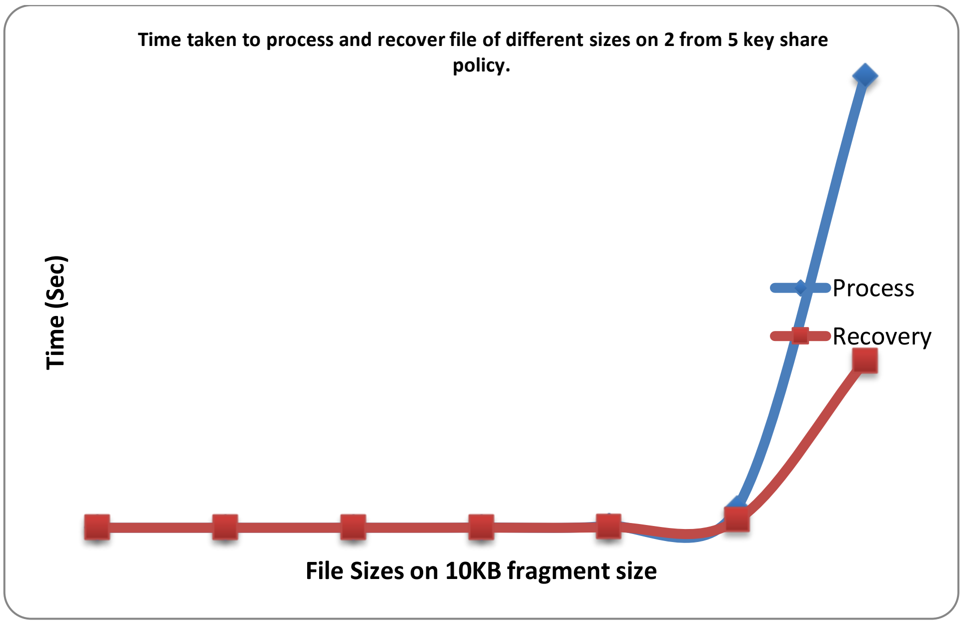


Figure 31: Process and Recover of file using 10KB fragment size on 2 from 5 Policy.

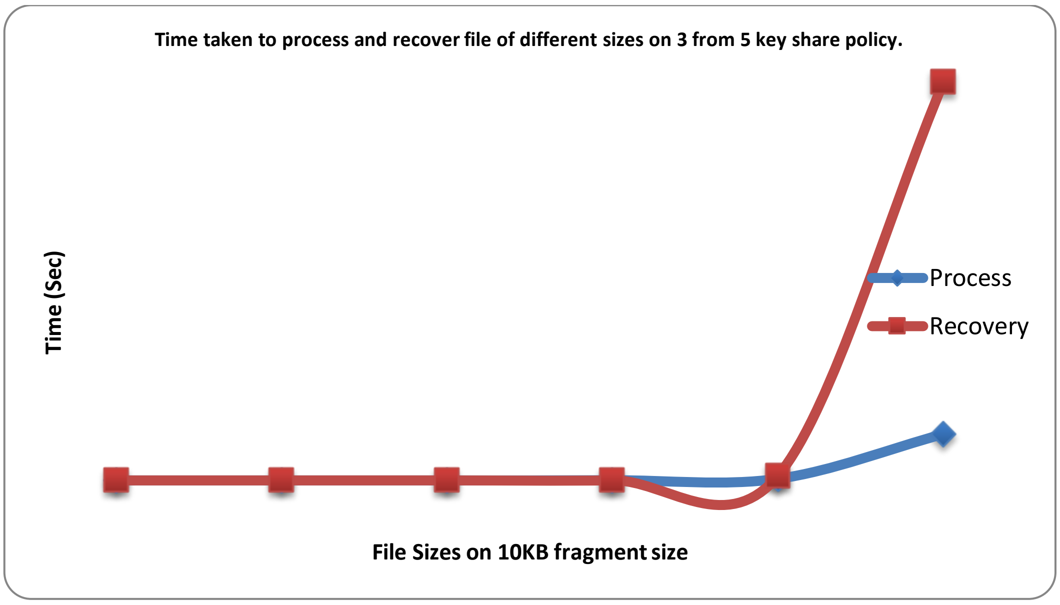


Figure 32: Process and Recover of file using 10KB fragment size on 3 from 5 Policy.

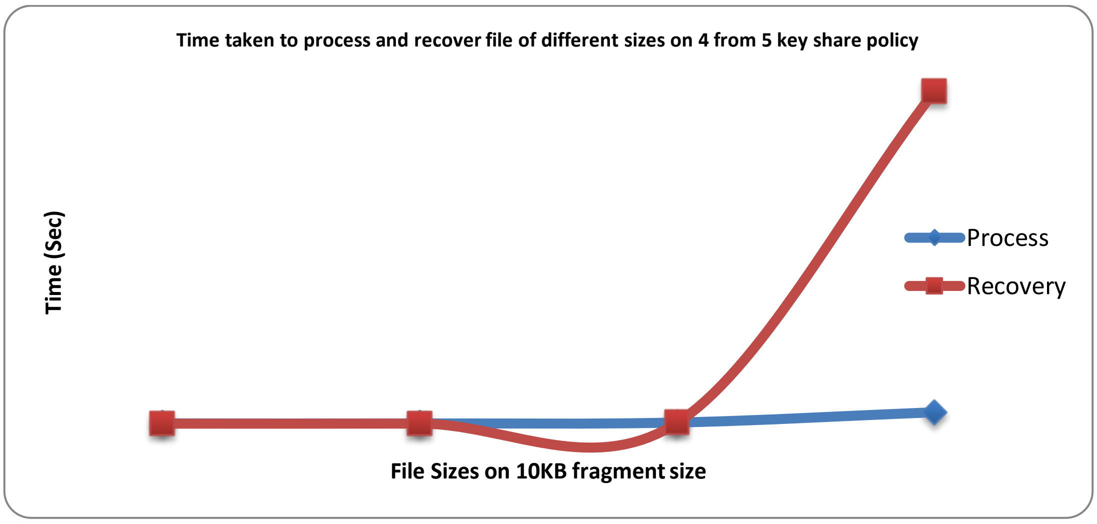


Figure 33: Process and Recover of file using 10KB fragment size on 4 from 5 Policy.

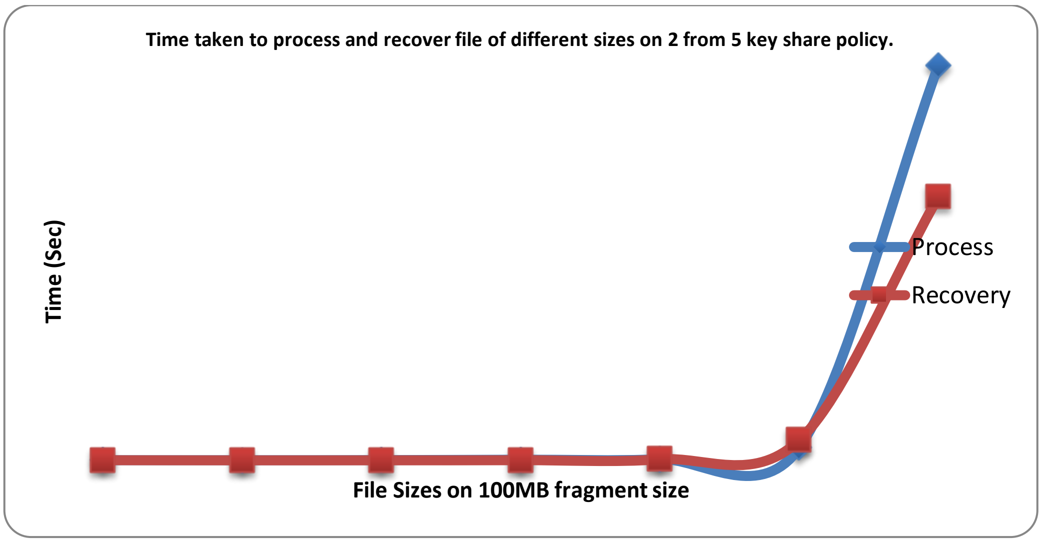


Figure 34: Process and Recover of file using 100MB fragment size on 2 from 5 Policy.

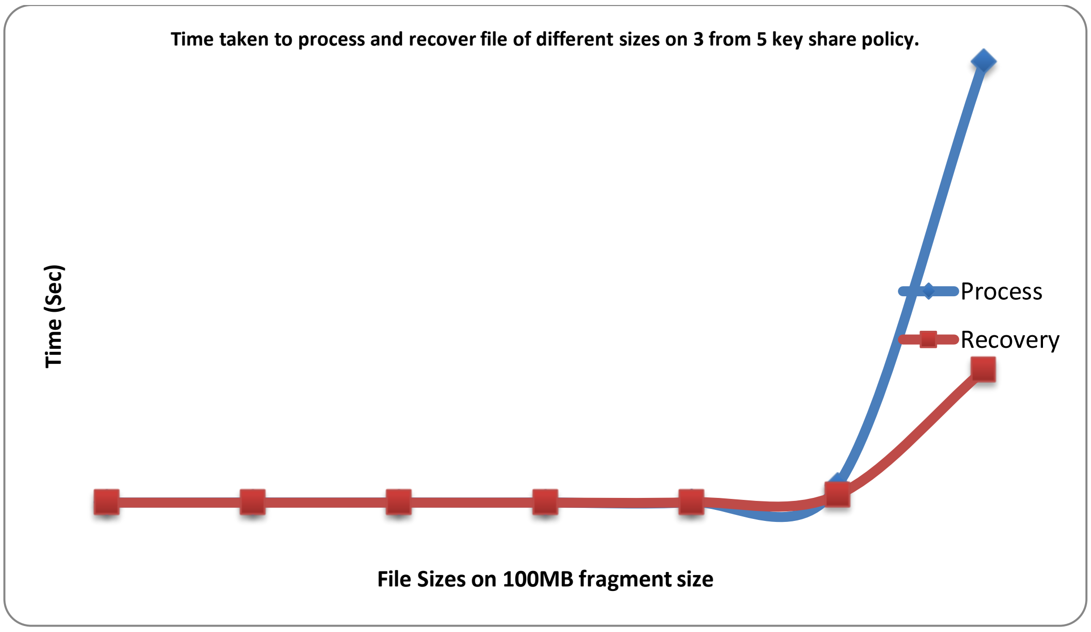


Figure 35: Process and Recover of a file using 100MB fragment size on 3 from 5 Policy.

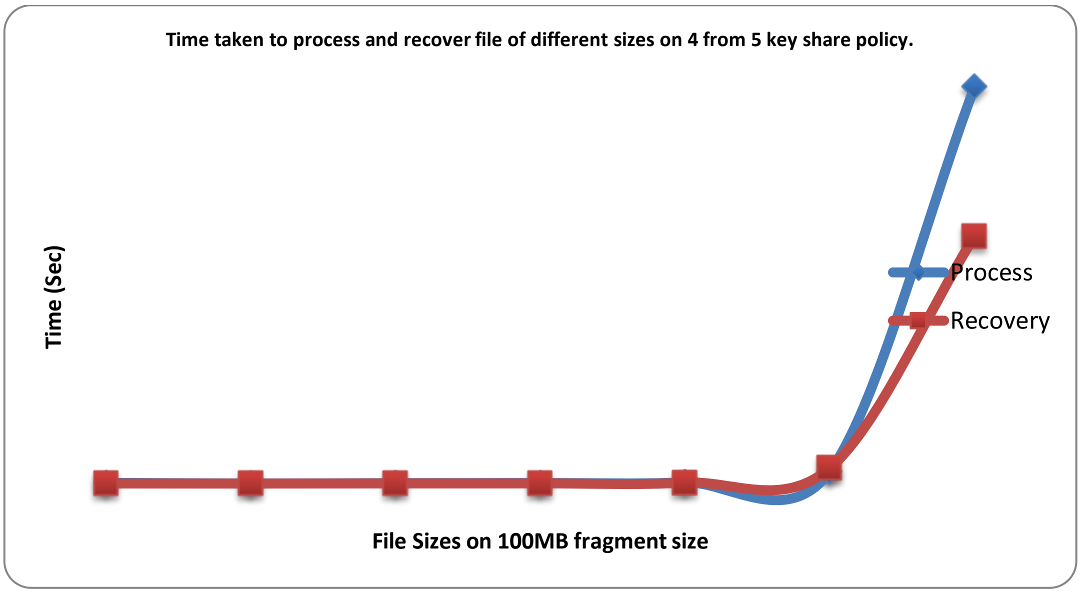


Figure 36: Process and Recover of a file using 100MB fragment size on 4 from 5 Policy.

# 11 Appendix B

## 11.1 Basic Data Striping and RAID Systems

RAID stands for Redundant Array of Independent Disks. It is a well-known method of combining several hard disk drives into one logical unit, so as to address the fault-tolerance and performance limitations of an individual hard disk drive. The key notions behind a RAID storage system are Data Striping and various levels of Redundancy. The striping process takes large data blocks and splits them into blocks of a defined size, such as 4 KB, which are then spread across each of the disks in the array. This can increase the performance of a storage system as a $n$-disk array (*i.e.*, $n$-way striping) provides $n$ times the read and write speed improvement of a single disk (although in practice the actual performance achieved tends to be less than $n$ times due to control overheads). However, the trade-off is reliability, as the failure of any individual disk would result in the failure of the entire array. Formally, assume that each independent hard disk drive has an identical rate of failure $r$, then the overall failure rate of an array of $n$ disks is:

$$1 - (1 - r)^n \qquad \textit{Equation 22}$$

Basic data striping over an array of $n$-disks without redundancy is referred to as RAID0, which is only suitable to situations where the highest I/O performance is desired, whilst the resulting increased probability of data loss can be tolerated. In other situations, reliability of data may outweigh system performance, and thus instead of striping the data over $n$ disks, the same data is duplicated (or mirrored) to $n$ disks. This strategy is referred to as RAID1, which is suitable to ensure the reliability of critical data. In comparison to RAID0, the overall failure rate of a RAID1 array of $n$ disks becomes:

$$r^n \qquad \textit{Equation 23}$$

However, when the same set of data is duplicated over $n$ disks, it would result in a storage overhead of $n - 1$ disks.

RAID0 and RAID1 can be combined into a RAID01 or a RAID10 configuration. The former means a mirrored configuration of multiple striped sets (*i.e.*, mirror of stripes); and the latter means a stripe across a number of mirrored sets (*i.e.*, stripe of mirrors). Both strategies are able to provide very good performance and reliability, whereas both are expensive solutions as considerable amount of hard disk storage is committed to maintaining redundancy information.

Another RAID level, namely RAID5, attempts to mitigate the high costs of RAID01/10. RAID5 utilizes striping and parity techniques to achieve simultaneously a similar I/O performance to RAID0 with a significantly lower failure rate of:

$$1 - (1 - r)^{n} - nr(1 - r)^{n-1} \qquad \textit{Equation 24}$$

A parity bit (or check bit) is a bit added to the end of a string of binary code that indicates whether the number of bits in the string with the value 1 is even or odd. In computing and telecommunications, a parity bit is calculated via an XOR sum of all the previous bits, yielding 0 for even parity and 1 for odd parity. Let A and B being two binary sequences. If:

X = A XOR B

then the following will be true:

A = X XOR B
B = X XOR A

Using this property, we can say that the following expressions are also true (and this can be repeated for an infinite amount of terms):

A XOR B XOR C XOR D = X

X XOR B XOR C XOR D = A

A XOR X XOR C XOR D = B

A XOR B XOR X XOR D = C

A XOR B XOR C XOR X = D

Put simply, if we calculate a XOR of a number sequence, we can substitute X for any number in this sequence, recalculate the XOR, and recover the original number. This is the principle for building a RAID5 array, of which an example is shown in Figure 37. Each of the four drives in Figure 37 are divided into four blocks, each belonging to a stripe on the same level across each drive. Each drive and each stripe have a parity block (stored in disk 4) which is the XOR of the other three blocks. So, say Disk 2 failed, then the XOR of A1 with A2 and A4 would get the remaining A3 for the A stripe; B1 with B2 and B3 to get the B4 parity block; *etc.* Generally speaking, by introducing a single parity bit, RAID5 is able to survive the failure of any single hard disk drive in any array which contains a minimum of three drives.

The RAID5 data striping mechanism can be applied to secret sharing of data in the Cloud, where each cloud storage provider represents a disk storing one share of the secret data, but the limitation of this lies in that it always requires $n - 1$ shares to reconstruct the original data, rather than a more desirable and flexible arbitrary *k*-out-of-*n* shares.

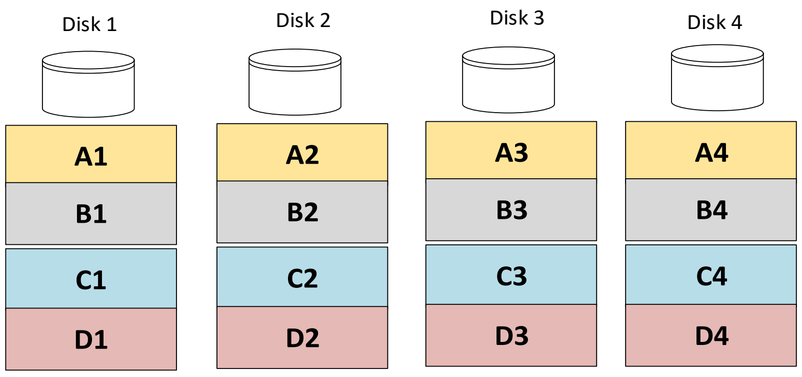


Figure 37: A 4-disk Array in RAID5 format

# 12 Appendix C

## 12.1 Detailed Experimental Results

## 12.2 Variant One

### 12.2.1 Fragments

**File Sizes: 1KB, 10KB, 100KB, 1MB, 10MB, 100MB, 1GB**

**Fragment Sizes: 1KB, 10KB, 100KB, 1MB, 10MB, 100MB, 1GB**

**Secret Sharing Policy: 2 from 5**

**Plot: File Sizes in KB against Time Taken in Seconds to process and combine file using a particular key share policy.**

Table 18: Varied file sizes using 1KB fragment size in 2 from 5 share policy

| S/N | FileSize | FileSplitTime | FragEncTime | FragDecTime | FileComTime | OverHeadCost |
|---|---|---|---|---|---|---|
| 1 | 1KB | 0.007814 | 0.019532 | 0.015503 | 0.011718 | 0.05456668 |
| 2 | 10KB | 0.046878 | 0.035514 | 0.467881 | 0.23438 | 0.78465236 |
| 3 | 100KB | 0.2394 | 0.32927 | 0.634978 | 0.209857 | 1.41350527 |
| 4 | 1MB | 2.128525 | 3.598536 | 5.165434 | 2.159326 | 13.0518216 |
| 5 | 10MB | 83.9537 | 129.7083 | 131.0154 | 59.06891 | 403.746322 |
| 6 | 100MB | 258.7115 | 3143.999 | 656.6972 | 1657.46 | 5716.86737 |

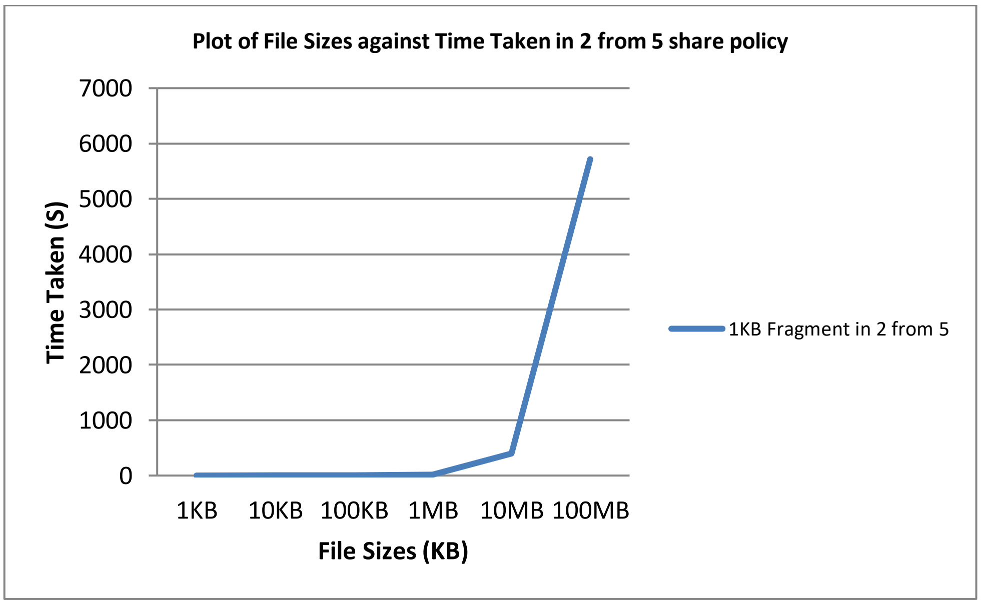


Figure 38: Varied file sizes using 1KB fragment size in 2 from 5 share policy

Table 19: Varied file sizes using 10KB fragment size in 2 from 5 share policy

| S/N | FileSize | FileSplitTime | FragEncTime | FragDecTime | FileComTime | OverHeadCost |
|---|---|---|---|---|---|---|
| 1 | 1KB | 0.015459 | 0.062503 | 0.007737 | 0.00761 | 0.09331036 |
| 2 | 10KB | 0.015662 | 0.027291 | 0 | 0.01556 | 0.05851249 |
| 3 | 100KB | 0.054534 | 0.07285 | 0.093732 | 0.054647 | 0.27576305 |
| 4 | 1MB | 0.383249 | 0.281557 | 0.564448 | 0.234383 | 1.46363745 |
| 5 | 10MB | 2.140715 | 2.6705 | 3.520125 | 2.093841 | 10.4251813 |
| 6 | 100MB | 22.28913 | 31.55624 | 38.52662 | 23.57686 | 115.948858 |
| 7 | 1GB | 385.7138 | 2098.181 | 905.8744 | 422.0202 | 3811.78945 |

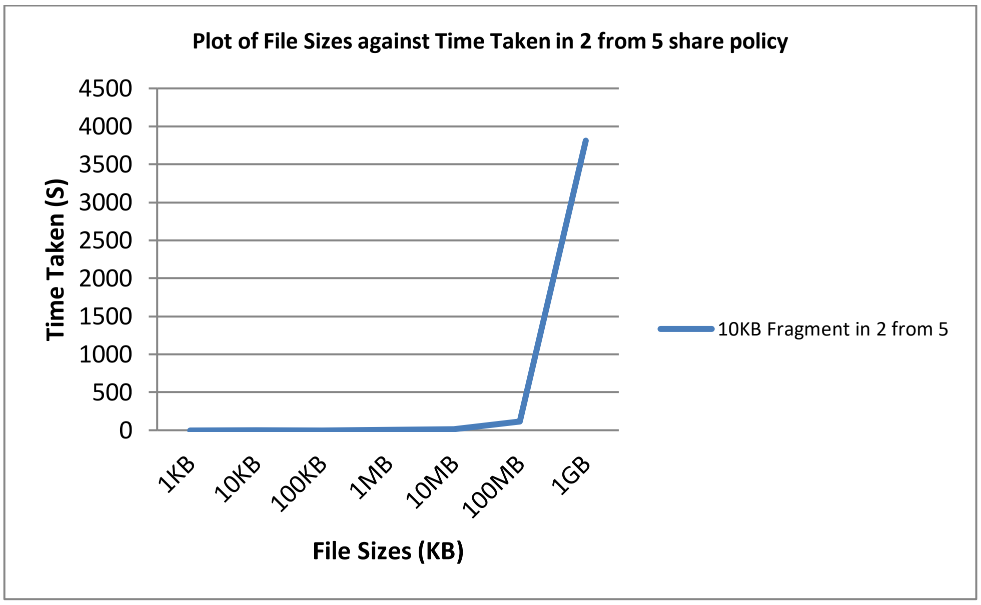


Figure 39: Varied file sizes using 10KB fragment size in 2 from 5 share policy

Table 20: Varied file sizes using 100KB fragment size in 2 from 5 share policy

| S/N | FileSize | FileSplitTime | FragEncTime | FragDecTime | FileComTime | OverHeadCost |
|---|---|---|---|---|---|---|
| 1 | 1KB | 0 | 0.031374 | 0.00774 | 0 | 0.039114 |
| 2 | 10KB | 0.015586 | 0.046881 | 0 | 0 | 0.06246698 |
| 3 | 100KB | 0.015625 | 0.011734 | 0.015587 | 0 | 0.04294586 |
| 4 | 1MB | 0.077863 | 0.120823 | 0.062505 | 0.031251 | 0.29244148 |
| 5 | 10MB | 0.468773 | 0.509873 | 0.695343 | 0.265636 | 1.93962426 |
| 6 | 100MB | 2.306734 | 4.857761 | 5.507886 | 4.238577 | 16.9109581 |
| 7 | 1GB | 65.50594 | 135.7359 | 136.9692 | 122.2016 | 460.412576 |

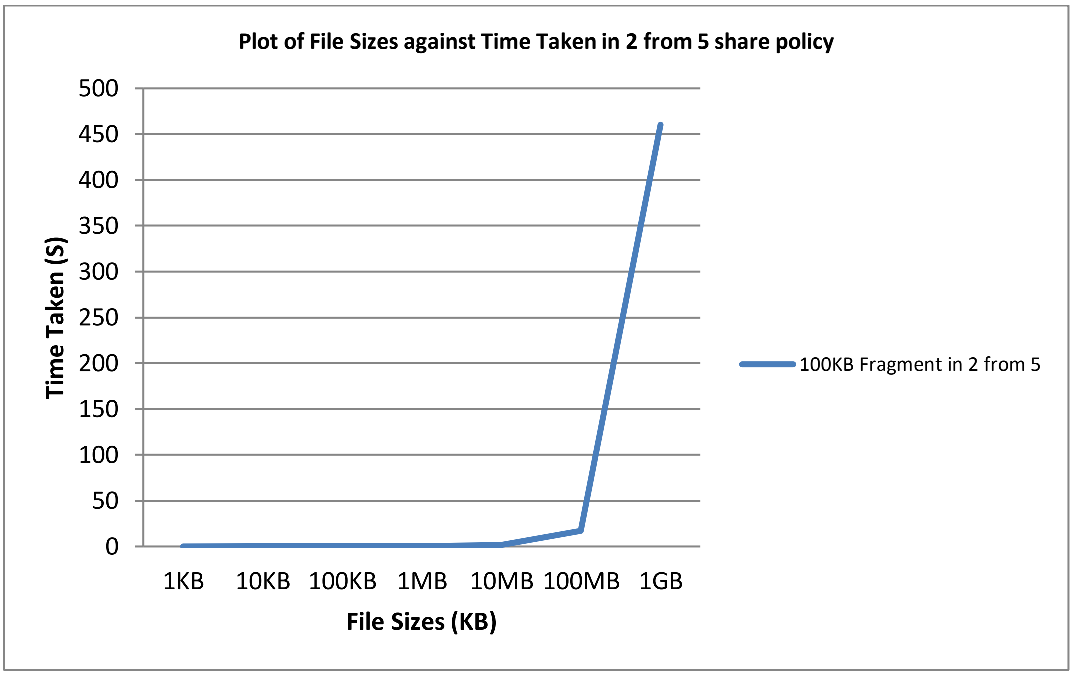


Figure 40: Varied file sizes using 100KB fragment size in 2 from 5 share policy

Table 21: Varied file sizes using 1MB fragment size in 2 from 5 share policy

| S/N | FileSize | FileSplitTime | FragEncTime | FragDecTime | FileComTime | OverHeadCost |
|---|---|---|---|---|---|---|
| 1 | 1KB | 0.01542 | 0.046879 | 0 | 0 | 0.06229949 |
| 2 | 10KB | 0.015629 | 0.039606 | 0.015504 | 0.007814 | 0.0785534 |
| 3 | 100KB | 0.007813 | 0.015626 | 0.015588 | 0 | 0.0390268 |
| 4 | 1MB | 0.026114 | 0.023432 | 0.039002 | 0.015626 | 0.10417414 |
| 5 | 10MB | 0.124867 | 0.364098 | 0.24216 | 0.054688 | 0.78581219 |
| 6 | 100MB | 0.715984 | 2.259576 | 2.371971 | 0.466472 | 5.81400332 |
| 7 | 1GB | 58.32339 | 100.316 | 133.5535 | 123.2983 | 415.491188 |

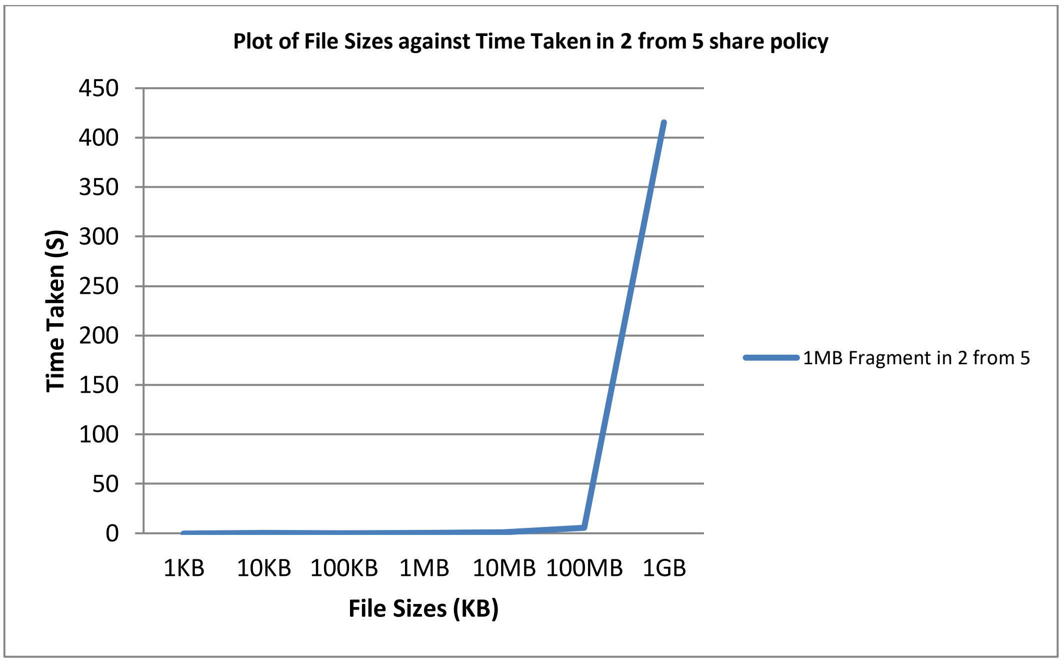


Figure 41: Varied file sizes using 1MB fragment size in 2 from 5 share policy

Table 22: Varied file sizes using 10MB fragment size in 2 from 5 share policy

| S/N | FileSize | FileSplitTime | FragEncTime | FragDecTime | FileComTime | OverHeadCost |
|---|---|---|---|---|---|---|
| 1 | 1KB | 0.023438 | 0.101659 | 0.109469 | 0.023346 | 0.2579124 |
| 2 | 10KB | 0.015488 | 0.406416 | 0.031252 | 0.015626 | 0.46878195 |
| 3 | 100KB | 0.01557 | 0.023438 | 0.015626 | 0.062598 | 0.11723244 |
| 4 | 1MB | 0.015625 | 0.03125 | 0.031253 | 0.007811 | 0.08593885 |
| 5 | 10MB | 0.804761 | 0.234951 | 1.023605 | 0.085927 | 2.14924483 |
| 6 | 100MB | 0.461024 | 1.729554 | 2.14624 | 0.321052 | 4.65786879 |
| 7 | 1GB | 43.87095 | 47.18494 | 64.00888 | 40.07847 | 195.143231 |

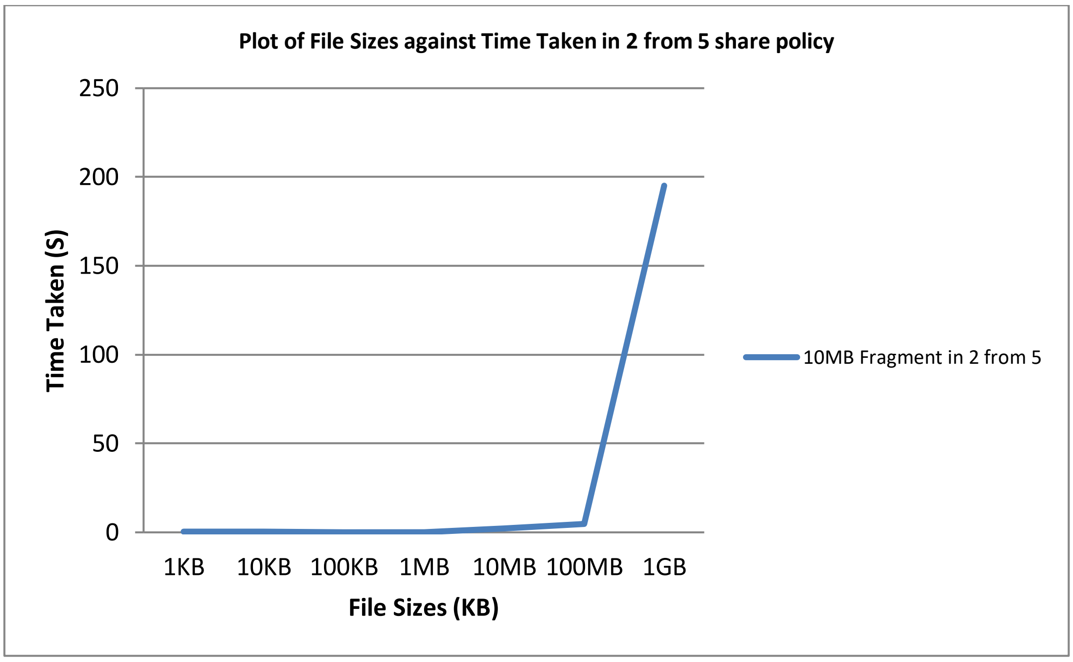


Figure 42: Varied file sizes using 10MB fragment size in 2 from 5 share policy

Table 23: Varied file sizes using 100MB fragment size in 2 from 5 share policy

| S/N | FileSize | FileSplitTime | FragEncTime | FragDecTime | FileComTime | OverHeadCost |
|---|---|---|---|---|---|---|
| 1 | 1KB | 0.007704 | 0.023322 | 0.031273 | 0 | 0.06229913 |
| 2 | 10KB | 0 | 0.03125 | 0.007814 | 0 | 0.03906357 |
| 3 | 100KB | 0.015669 | 0.031253 | 0.015565 | 0 | 0.06248677 |
| 4 | 1MB | 0.031139 | 0.086004 | 0.031161 | 0.007814 | 0.15611708 |
| 5 | 10MB | 0.046877 | 0.195385 | 0.195246 | 0.015627 | 0.45313442 |
| 6 | 100MB | 1.337963 | 1.310698 | 2.81248 | 1.437562 | 6.89870221 |
| 7 | 1GB | 55.83863 | 27.30553 | 34.04062 | 21.6859 | 138.870675 |

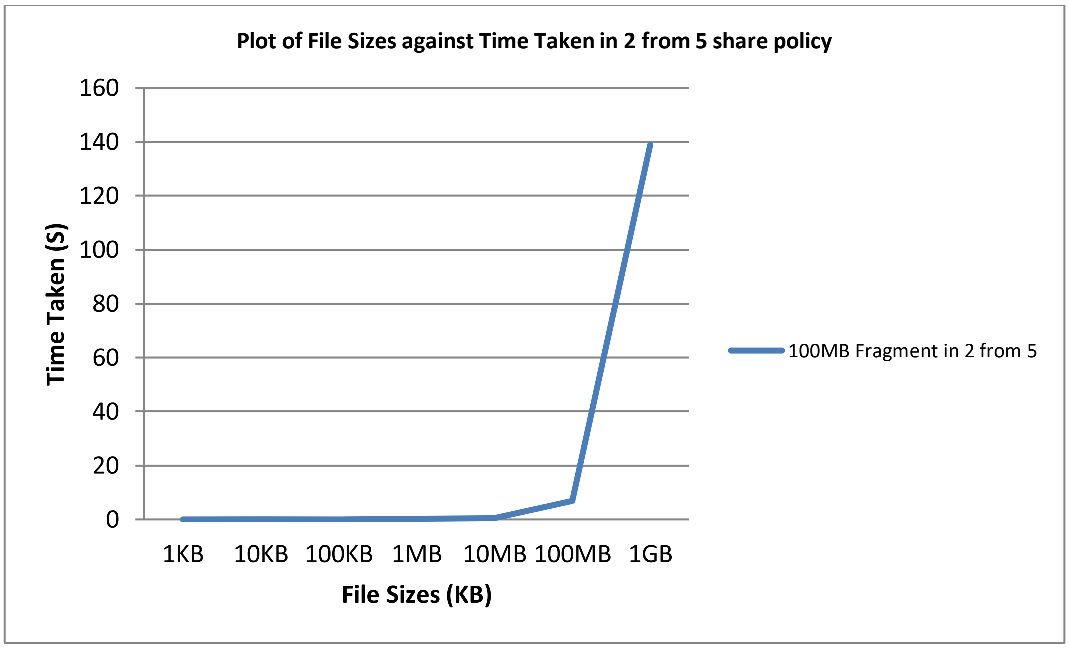


Figure 43: Varied file sizes using 100MB fragment size in 2 from 5 share policy

Table 24: Varied file sizes using 1GB fragment size in 2 from 5 share policy

| S/N | FileSize | FileSplitTime | FragEncTime | FragDecTime | FileComTime | OverHeadCost |
|---|---|---|---|---|---|---|
| 1 | 1KB | 0.007794 | 0.039104 | 0.015493 | 0 | 0.06239057 |
| 2 | 10KB | 0 | 0 | 0.023367 | 0 | 0.02336657 |
| 3 | 100KB | 0.009818 | 0.021634 | 0.023639 | 0.005499 | 0.06059051 |
| 4 | 1MB | 0.00801 | 0.038056 | 0.036052 | 0.005508 | 0.08762646 |
| 5 | 10MB | 0.102186 | 0.234294 | 0.218722 | 0.039147 | 0.59434903 |
| 6 | 100MB | 1.18749 | 2.782697 | 2.881051 | 1.257869 | 8.10910678 |
| 7 | 1GB | 39.4435 | 62.98048 | 434.7571 | 35.47601 | 572.657097 |

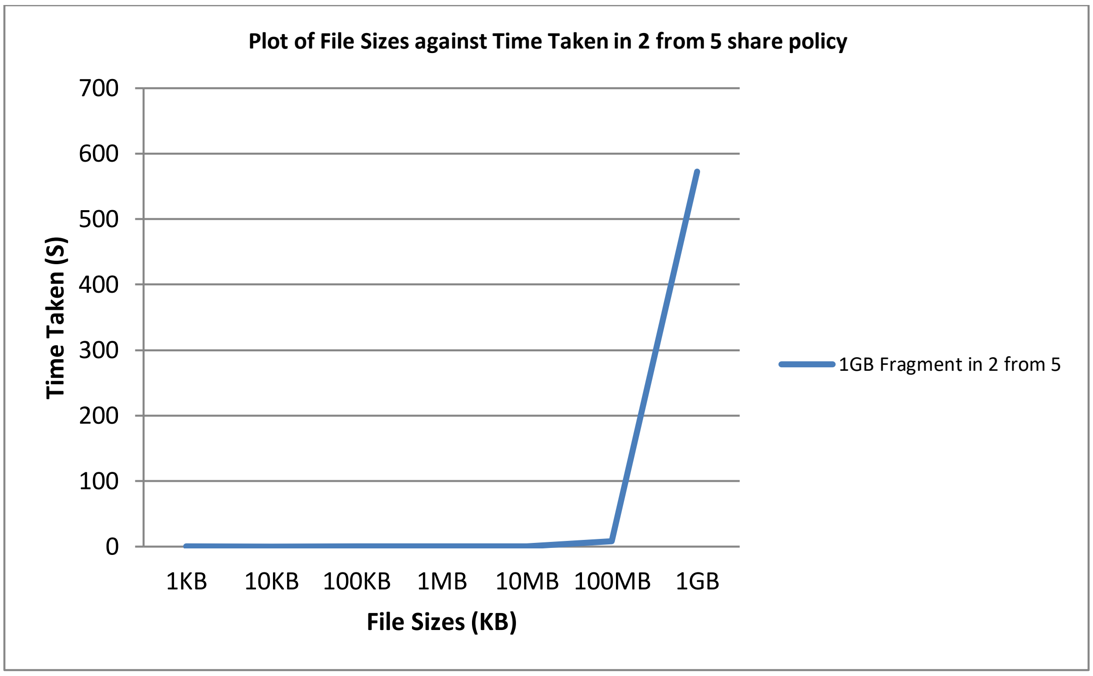


Figure 44: Varied file sizes using 1GB fragment size in 2 from 5 share policy

**File Sizes: 1KB, 10KB, 100KB, 1MB, 10MB, 100MB, 1GB**

**Fragment Sizes: 1KB, 10KB, 100KB, 1MB, 10MB, 100MB, 1GB**

**Secret Sharing Policy: 3 from 5**

**Plot: File Sizes in KB against Time Taken in Seconds to process and combine file using a particular key share policy.**

Table 25: Varied file sizes using 1 KB fragment size in 3 from 5 share policy

| S/N | FileSize | FileSplitTime | FragEncTime | FragDecTime | FileComTime | OverHeadCost |
|---|---|---|---|---|---|---|
| 1 | 1KB | 0.007814 | 0.024441 | 0.015626 | 0 | 0.04788077 |
| 2 | 10KB | 0.039009 | 0.028408 | 0.17977 | 0.038987 | 0.286173698 |
| 3 | 100KB | 0.355546 | 0.290273 | 0.507835 | 0.226573 | 1.380226719 |
| 4 | 1MB | 2.460981 | 2.749018 | 3.411468 | 1.961024 | 10.58249097 |
| 5 | 10MB | 20.73167 | 31.34608 | 71.71715 | 19.70122 | 143.496118 |

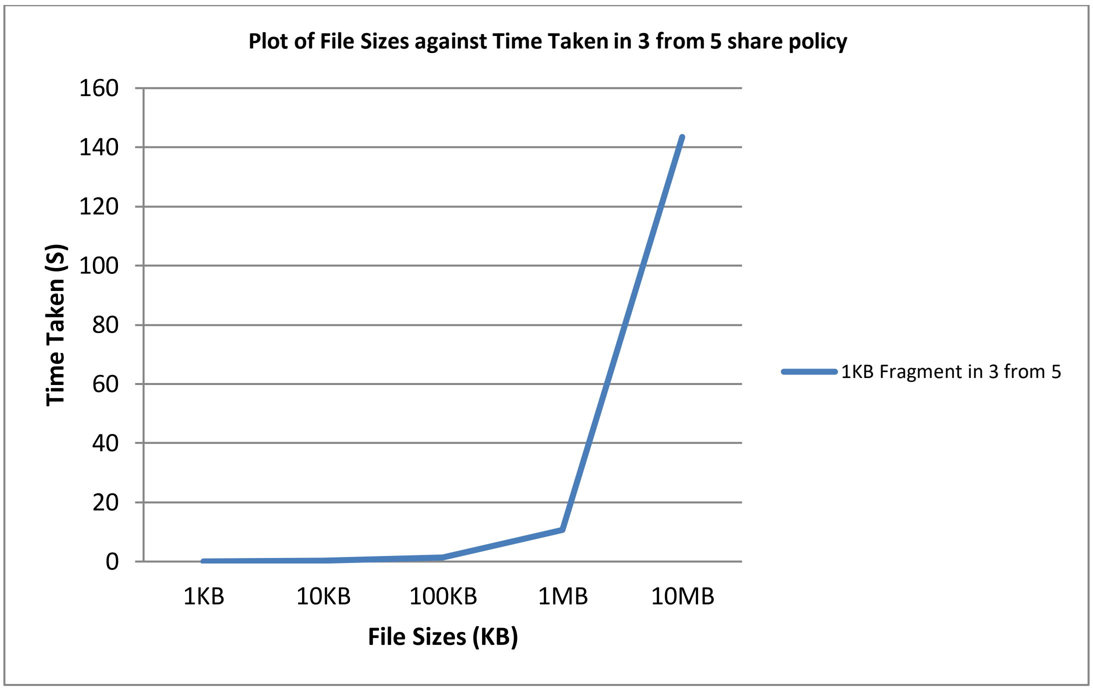


Figure 45: Varied file sizes using 1KB fragment size in 3 from 5 share policy

Table 26: Varied file sizes using 10KB fragment size in 3 from 5 share policy

| S/N | FileSize | FileSplitTime | FragEncTime | FragDecTime | FileComTime | OverHeadCost |
|---|---|---|---|---|---|---|
| 1 | 1KB | 0.007815 | 0.054611 | 0.007871 | 0 | 0.070297003 |
| 2 | 10KB | 0.015624 | 0.023468 | 0.031187 | 0.007814 | 0.078092813 |
| 3 | 100KB | 0.070212 | 0.05682 | 0.062431 | 0.023438 | 0.21290154 |
| 4 | 1MB | 0.523388 | 0.386838 | 0.507764 | 0.250011 | 1.668001673 |
| 5 | 10MB | 3.164124 | 2.763088 | 7.033319 | 4.414262 | 17.37479342 |
| 6 | 100MB | 96.407 | 93.02779 | 212.402 | 148.634 | 550.4707818 |

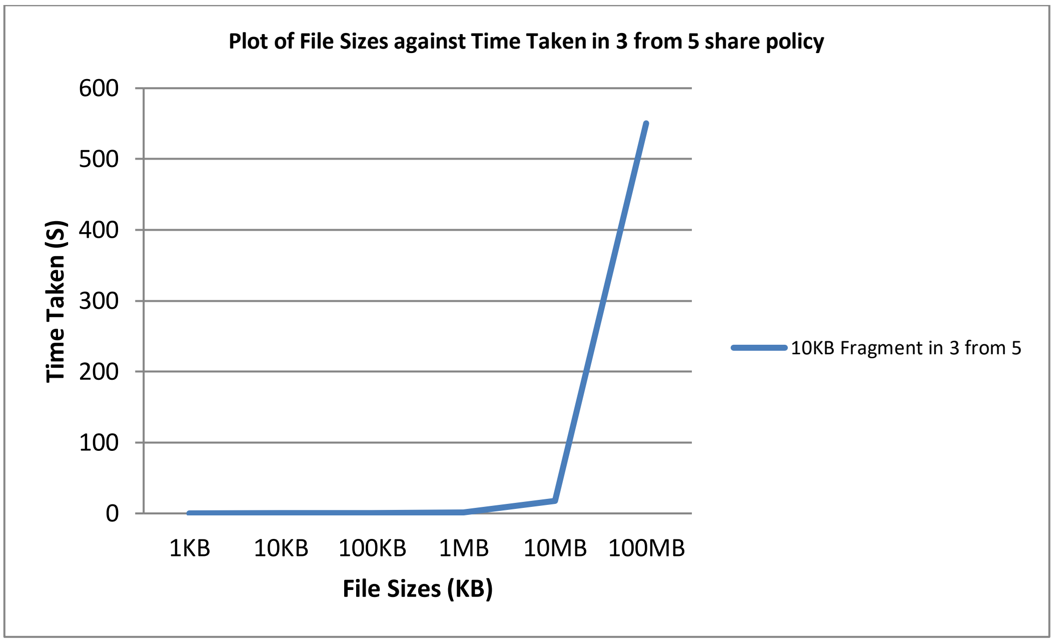


Figure 46: Varied file sizes using 10KB fragment size in 3 from 5 share policy

Table 27: Varied file sizes using 100KB fragment size in 3 from 5 share policy

| S/N | FileSize | FileSplitTime | FragEncTime | FragDecTime | FileComTime | OverHeadCost |
|---|---|---|---|---|---|---|
| 1 | 1KB | 0.032046 | 0.055771 | 0.008948 | 0.008453 | 0.10521841 |
| 2 | 10KB | 0.035926 | 0.02375 | 0.030348 | 0.074121 | 0.164145291 |
| 3 | 100KB | 0.085691 | 0.007237 | 0.096141 | 0.450164 | 0.639233004 |
| 4 | 1MB | 0.386582 | 0.059073 | 0.739658 | 2.010074 | 3.195386351 |
| 5 | 10MB | 5.276726 | 0.590746 | 7.817365 | 3.073143 | 16.75798046 |
| 6 | 100MB | 51.49968 | 5.967706 | 211.5395 | 30.44199 | 299.448869 |

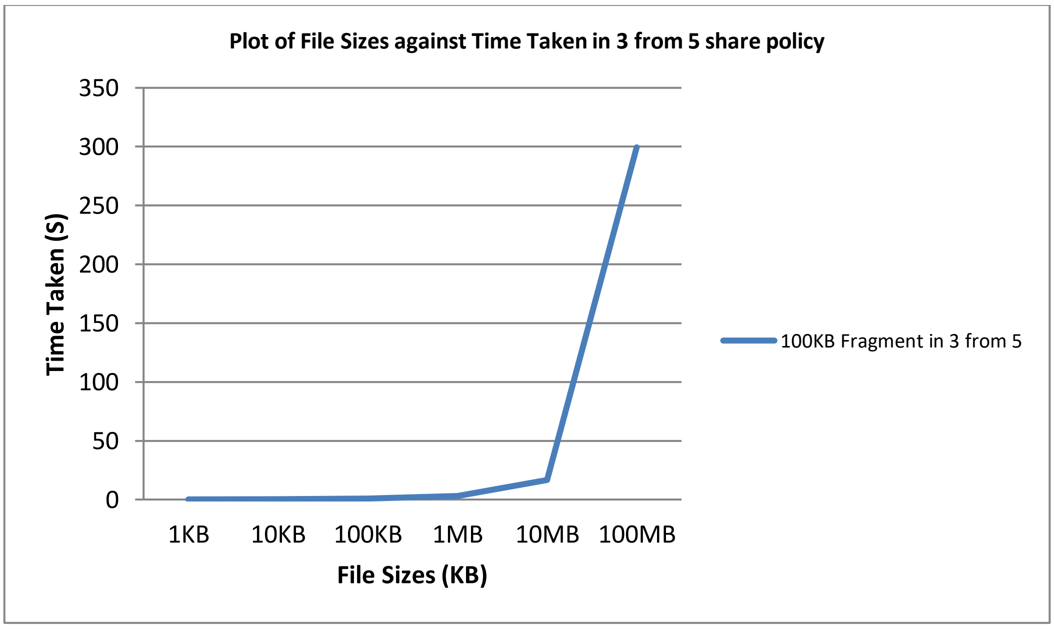


Figure 47: Varied file sizes using 100KB fragment size in 3 from 5 share policy

Table 28: Varied file sizes using 1MB fragment size in 3 from 5 share policy

| S/N | FileSize | FileSplitTime | FragEncTime | FragDecTime | FileComTime | OverHeadCost |
|---|---|---|---|---|---|---|
| 1 | 1KB | 0.008512 | 0.065378 | 0.008515 | 0.035585 | 0.117990375 |
| 2 | 10KB | 0.01302 | 0.02103 | 0.010015 | 0.014521 | 0.058585167 |
| 3 | 100KB | 0.021531 | 0.023533 | 0.011017 | 0.025536 | 0.081617594 |
| 4 | 1MB | 0.017026 | 0.065158 | 0.030042 | 0.031525 | 0.143750549 |
| 5 | 10MB | 0.108657 | 0.236346 | 0.303846 | 0.056082 | 0.704930187 |
| 6 | 100MB | 1.344405 | 3.045307 | 3.22482 | 0.691046 | 8.305577577 |
| 7 | 1GB | 104.6643 | 49.06252 | 62.80052 | 37.64411 | 254.171503 |

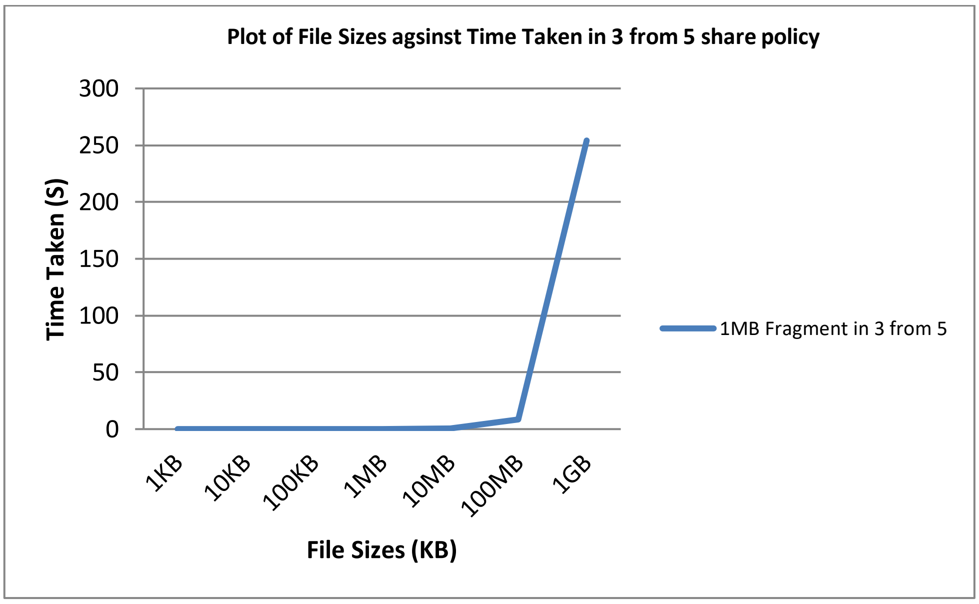


Figure 48: Varied file sizes using 1MB fragment size in 3 from 5 share policy

Table 29: Varied file sizes using 10MB fragment size in 3 from 5 share policy

| S/N | FileSize | FileSplitTime | FragEncTime | FragDecTime | FileComTime | OverHeadCost |
|---|---|---|---|---|---|---|
| 1 | 1KB | 0.020112 | 0.023535 | 0.009015 | 0.007514 | 0.060176014 |
| 2 | 10KB | 0.026537 | 0.036577 | 0.009023 | 0.007011 | 0.079147697 |
| 3 | 100KB | 0.01502 | 0.033179 | 0.011017 | 0.022862 | 0.082077845 |
| 4 | 1MB | 0.023534 | 0.034552 | 0.036846 | 0.025393 | 0.120325445 |
| 5 | 10MB | 0.058586 | 0.191281 | 0.199291 | 0.030044 | 0.479201556 |
| 6 | 100MB | 0.499753 | 1.877582 | 1.954264 | 0.262493 | 4.594092017 |
| 7 | 1GB | 96.12252 | 43.16561 | 53.37338 | 37.46342 | 230.1249373 |

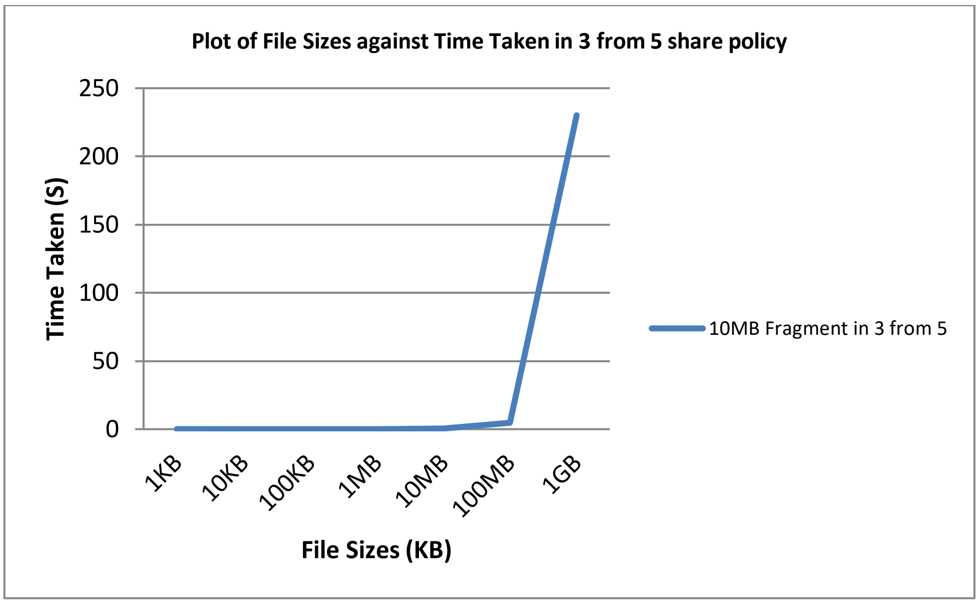


Figure 49: Varied file sizes using 10MB fragment size in 3 from 5 share policy

Table 30: Varied file sizes using 100MB fragment size in 3 from 5 share policy

| S/N | FileSize | FileSplitTime | FragEncTime | FragDecTime | FileComTime | OverHeadCost |
|---|---|---|---|---|---|---|
| 1 | 1KB | 0.019525 | 0.034548 | 0.009015 | 0.032547 | 0.095635414 |
| 2 | 10KB | 0.031044 | 0.047069 | 0.009012 | 0.007512 | 0.094637393 |
| 3 | 100KB | 0.028545 | 0.046119 | 0.030143 | 0.007513 | 0.112320304 |
| 4 | 1MB | 0.028566 | 0.061591 | 0.030044 | 0.009013 | 0.129214049 |
| 5 | 10MB | 0.078111 | 0.204799 | 0.194784 | 0.051771 | 0.529465198 |
| 6 | 100MB | 1.864599 | 7.377933 | 3.008147 | 1.185042 | 13.43572044 |
| 7 | 1GB | 194.9306 | 39.19965 | 42.25581 | 28.53763 | 304.9236548 |

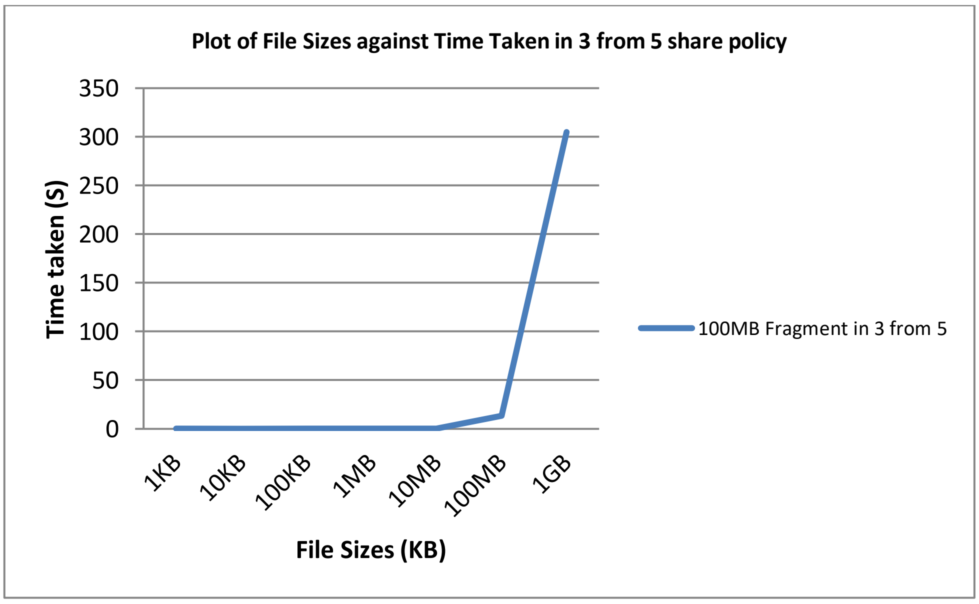


Figure 50: Varied file sizes using 100MB fragment size in 3 from 5 share policy

Table 31: Varied file sizes using 1GB fragment size in 3 from 5 share policy

| S/N | FileSize | FileSplitTime | FragEncTime | FragDecTime | FileComTime | OverHeadCost |
|---|---|---|---|---|---|---|
| 1 | 1KB | 0.012519 | 0.02003 | 0.021626 | 0.007512 | 0.06168723 |
| 2 | 10KB | 0.014022 | 0.025037 | 0.028429 | 0.008012 | 0.075499296 |
| 3 | 100KB | 0.024529 | 0.026193 | 0.01197 | 0.00751 | 0.070202584 |
| 4 | 1MB | 0.010514 | 0.042563 | 0.031548 | 0.010013 | 0.094639063 |
| 5 | 10MB | 0.055076 | 0.187779 | 0.195324 | 0.038056 | 0.476234913 |
| 6 | 100MB | 1.785259 | 2.842892 | 3.164258 | 1.471594 | 9.264002561 |
| 7 | 1GB | 223.3865 | 572.8175 | 1050.397 | 322.2105 | 2168.811955 |

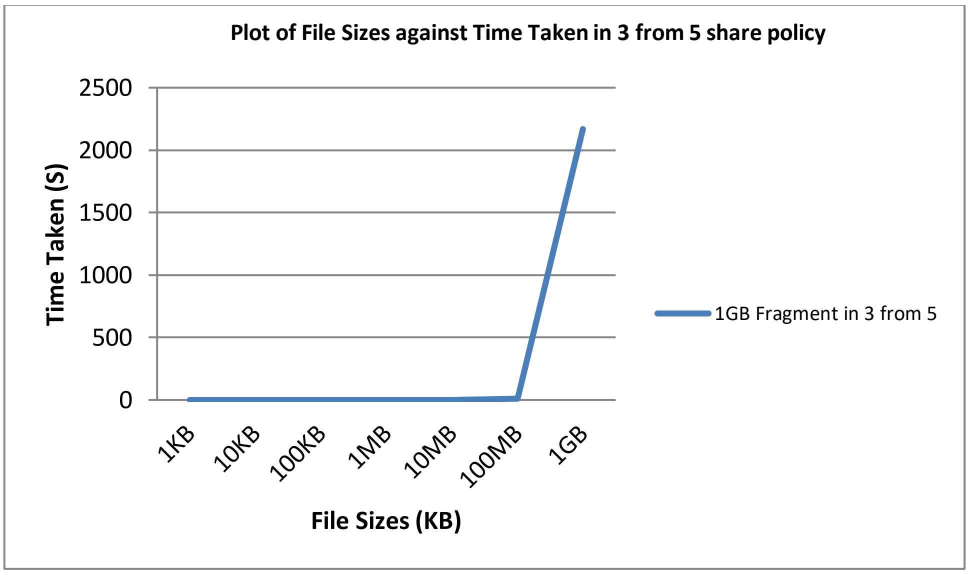


Figure 51: Varied file sizes using 1GB fragment size in 3 from 5 share policy

**File Sizes: 1KB, 10KB, 100KB, 1MB, 10MB, 100MB, 1GB**

**Fragment Sizes: 1KB, 10KB, 100KB, 1MB, 10MB, 100MB, 1GB**

**Secret Sharing Policy: 4 from 5**

**Plot: File Sizes in KB against Time Taken in Seconds to process and combine file using a particular key share policy.**

Table 32: Varied file sizes using 1KB fragment size in 4 from 5 share policy

| S/N | FileSize | FileSplitTime | FragEncTime | FragDecTime | FileComTime | OverHeadCost |
|---|---|---|---|---|---|---|
| 1 | 1KB | 0.045069 | 0.017524 | 0.04006 | 0.022031 | 0.124684453 |
| 2 | 10KB | 0.089797 | 0.063733 | 0.205464 | 0.040059 | 0.399052507 |
| 3 | 100KB | 0.441522 | 0.552298 | 1.030704 | 0.331485 | 2.356008299 |

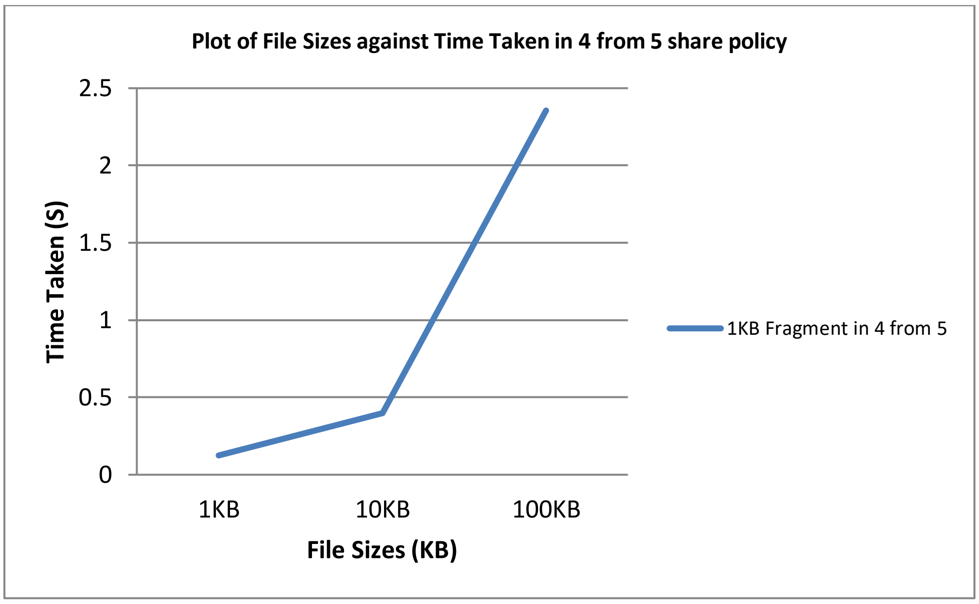


Figure 52: Varied file sizes using 1KB fragment size in 4 from 5 share policy

Table 33: Varied file sizes using 10KB fragment size in 4 from 5 share policy

| S/N | FileSize | FileSplitTime | FragEncTime | FragDecTime | FileComTime | OverHeadCost |
|---|---|---|---|---|---|---|
| 1 | 1KB | 0.054735 | 0.029455 | 0.010021 | 0.00801 | 0.102221012 |
| 2 | 10KB | 0.052071 | 0.023409 | 0.018026 | 0.011017 | 0.104522109 |
| 3 | 100KB | 0.06436 | 0.06374 | 0.124244 | 0.041061 | 0.293404951 |
| 4 | 1MB | 0.497475 | 0.561048 | 0.923853 | 0.394741 | 2.377116688 |

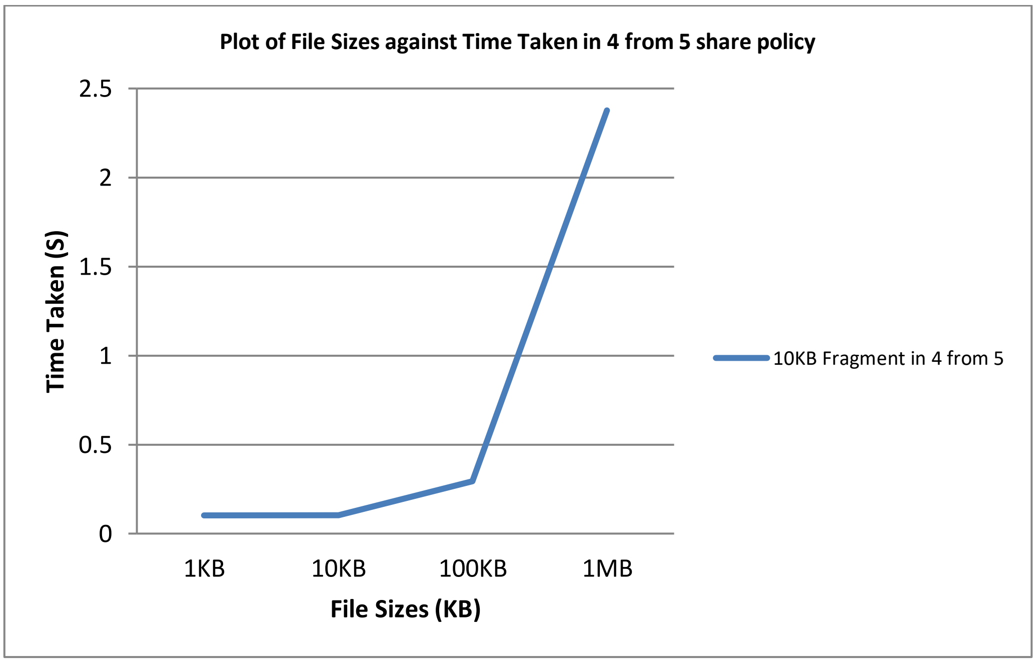


Figure 53: Varied file sizes using 10KB fragment size in 4 from 5 share policy

Table 34: Varied file sizes using 100KB fragment size in 4 from 5 share policy

| S/N | FileSize | FileSplitTime | FragEncTime | FragDecTime | FileComTim e | OverHeadCost |
|---|---|---|---|---|---|---|
| 1 | 1KB | 0.011017 | 0.016023 | 0.010015 | 0.008013 | 0.045068264 |
| 2 | 10KB | 0.016021 | 0.020027 | 0.011017 | 0.080083 | 0.12714839 |
| 3 | 100KB | 0.033045 | 0.017527 | 0.019031 | 0.011016 | 0.0806185 |
| 4 | 1MB | 0.076108 | 0.076481 | 0.151044 | 0.042059 | 0.345691375 |
| 5 | 10MB | 0.489383 | 1.046031 | 0.995171 | 0.389736 | 2.920321628 |

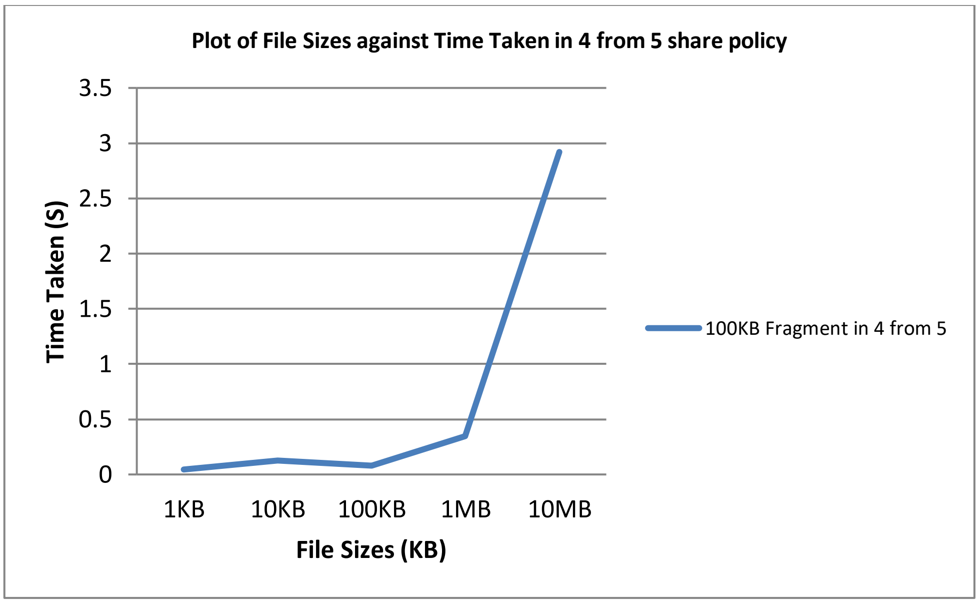


Figure 54: Varied file sizes using 100KB fragment size in 4 from 5 share policy

Table 35: Varied file sizes using 1MB fragment size in 4 from 5 share policy

| S/N | FileSize | FileSplitTime | FragEncTime | FragDecTime | FileComTime | OverHeadCost |
|---|---|---|---|---|---|---|
| 1 | 1KB | 0.015165 | 0.019025 | 0.009014 | 0.007011 | 0.050215244 |
| 2 | 10KB | 0.016024 | 0.019028 | 0.010012 | 0.00801 | 0.05307436 |
| 3 | 100KB | 0.009013 | 0.01903 | 0.012019 | 0.008011 | 0.048072815 |
| 4 | 1MB | 0.032046 | 0.070643 | 0.039058 | 0.036055 | 0.177802086 |
| 5 | 10MB | 0.242368 | 0.28314 | 0.288423 | 0.082094 | 0.896025287 |
| 6 | 100MB | 1.427057 | 3.188541 | 3.232013 | 0.870321 | 8.71793245 |

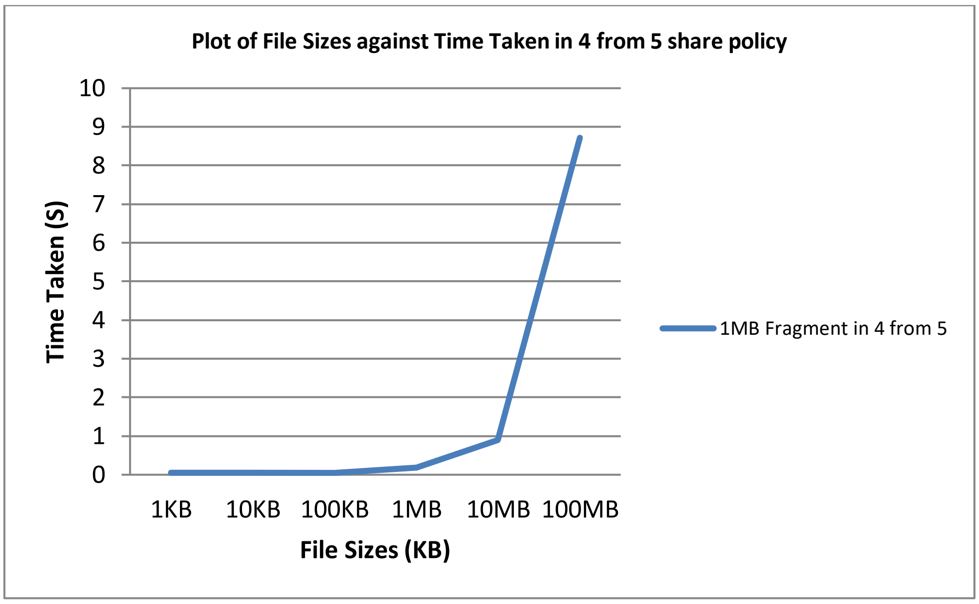


Figure 55: Varied file sizes using 1MB fragment size in 4 from 5 share policy

Table 36: Varied file sizes using 10MB fragment size in 4 from 5 share policy

| S/N | FileSize | FileSplitTime | FragEncTime | FragDecTime | FileComTime | OverHeadCost |
|---|---|---|---|---|---|---|
| 1 | 1KB | 0.021032 | 0.079934 | 0.011017 | 0.007009 | 0.118992329 |
| 2 | 10KB | 0.027039 | 0.016024 | 0.009015 | 0.00701 | 0.059087992 |
| 3 | 100KB | 0.033048 | 0.076109 | 0.012018 | 0.039436 | 0.160611153 |
| 4 | 1MB | 0.03004 | 0.076736 | 0.032045 | 0.010015 | 0.148836136 |
| 5 | 10MB | 0.098146 | 0.179262 | 0.279647 | 0.071104 | 0.628159046 |
| 6 | 100MB | 0.818294 | 2.268259 | 2.388486 | 0.403592 | 5.878630726 |
| 7 | 1GB | 87.14479 | 49.09185 | 49.53975 | 41.1374 | 226.9137902 |

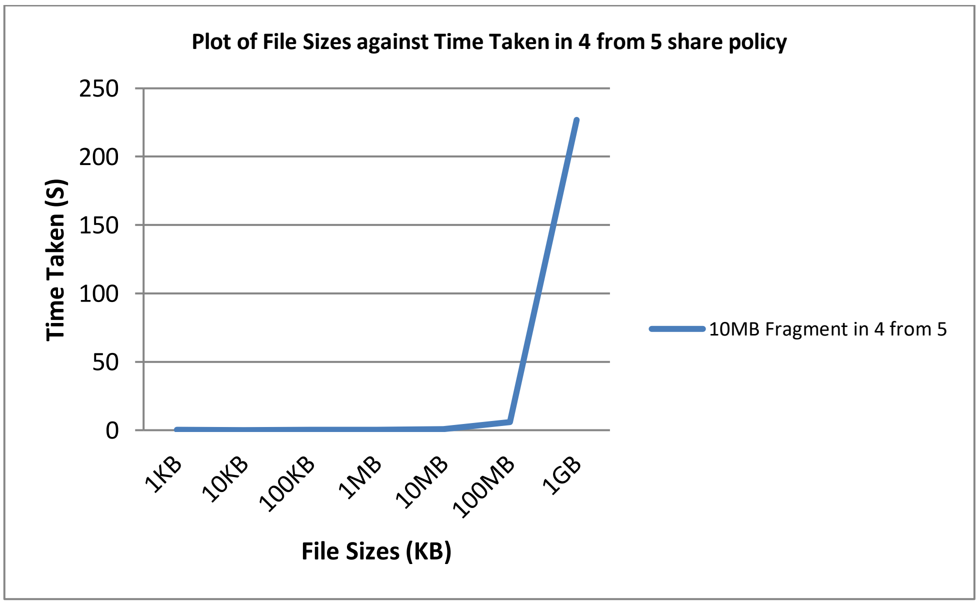


Figure 56: Varied file sizes using 10MB fragment size in 4 from 5 share policy

Table 37: Varied file sizes using 100MB fragment size in 4 from 5 share policy

| S/N | FileSize | FileSplitTime | FragEncTime | FragDecTime | FileComTime | OverHeadCost |
|---|---|---|---|---|---|---|
| 1 | 1KB | 0.054091 | 0.078141 | 0.010017 | 0.008011 | 0.150259972 |
| 2 | 10KB | 0.024037 | 0.017024 | 0.008889 | 0.008013 | 0.057962894 |
| 3 | 100KB | 0.022032 | 0.067101 | 0.032055 | 0.022034 | 0.143222332 |
| 4 | 1MB | 0.044828 | 0.080118 | 0.031864 | 0.026553 | 0.183362961 |
| 5 | 10MB | 0.088123 | 0.224328 | 0.231341 | 0.028037 | 0.571829319 |
| 6 | 100MB | 1.746907 | 1.490762 | 3.070401 | 1.048018 | 7.356088161 |
| 7 | 1GB | 73.1758 | 31.97497 | 37.34199 | 28.54653 | 171.0392945 |

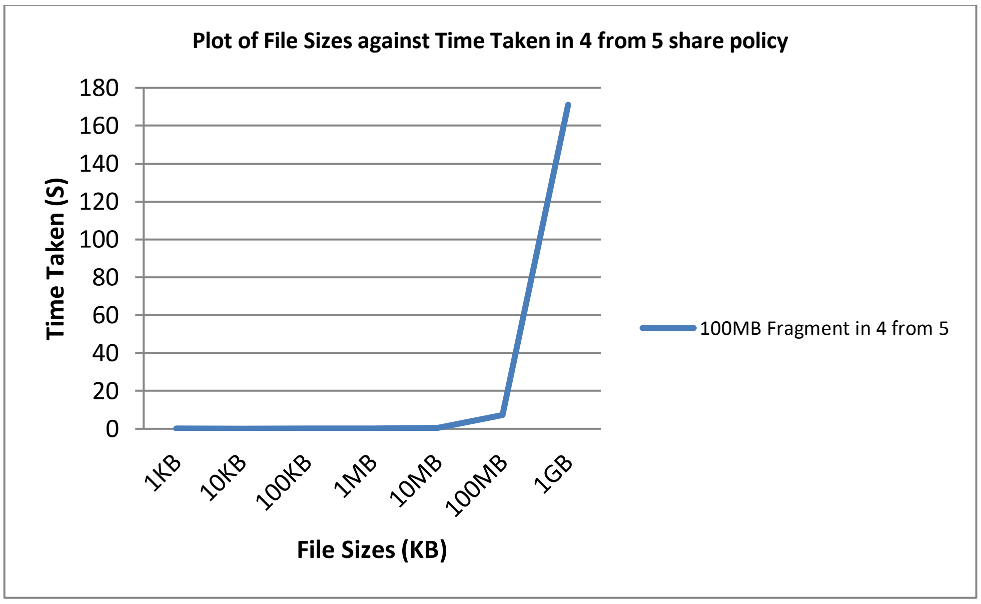


Figure 57: Varied file sizes using 100MB fragment size in share policy

Table 38: Varied file sizes using 1GB fragment size in 4 from 5 share policy

| S/N | FileSize | FileSplitTime | FragEncTime | FragDecTime | FileComTime | OverHeadCost |
|---|---|---|---|---|---|---|
| 1 | 1KB | 0.021034 | 0.166063 | 0.010013 | 0.007008 | 0.204117775 |
| 2 | 10KB | 0.021028 | 0.038056 | 0.011017 | 0.008012 | 0.078113079 |
| 3 | 100KB | 0.045064 | 0.058084 | 0.047072 | 0.007009 | 0.157228947 |
| 4 | 1MB | 0.052076 | 0.121175 | 0.051073 | 0.010013 | 0.23433733 |
| 5 | 10MB | 0.081123 | 0.203299 | 0.194283 | 0.030045 | 0.508749962 |
| 6 | 100MB | 1.876107 | 3.978742 | 3.320234 | 1.230602 | 10.40568471 |
| 7 | 1GB | 87.97622 | 562.5093 | 1627.002 | 38.63254 | 2316.119604 |

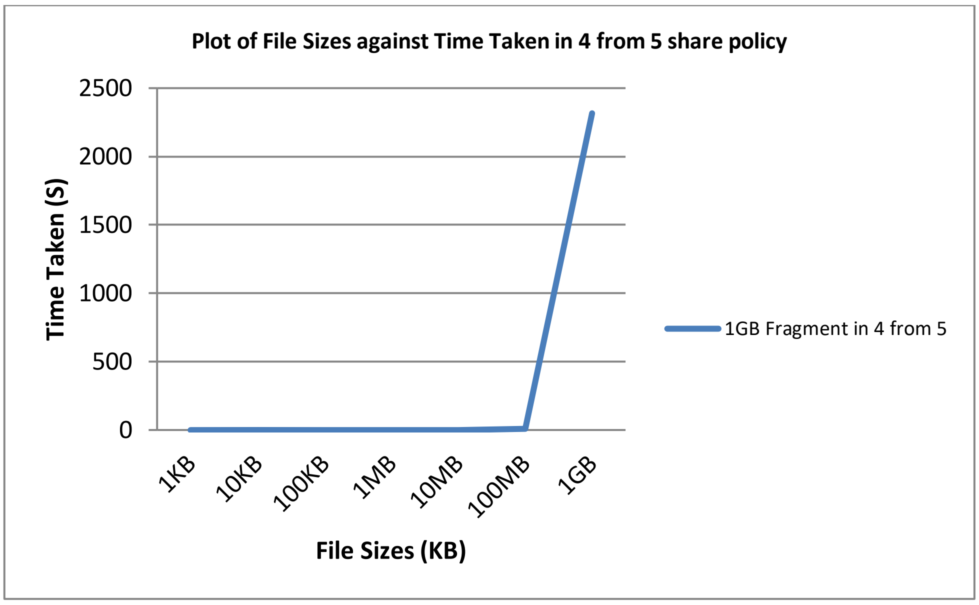


Figure 58: Varied file sizes using 1GB fragment size in 4 from 5 share policy

### 12.2.2 Key Share Creation and Recovering

**File Sizes: 1KB, 10KB, 100KB, 1MB, 10MB, 100MB, 1GB**

**Fragment Sizes: 1KB, 10KB, 100KB, 1MB, 10MB, 100MB, 1GB**

**Secret Sharing Policy: 2 from 5**

**Plot: File Sizes in KB against Time Taken in Seconds to process and recover secret key using a particular key share policy.**

Table 39: Key Share Creation and Recovering using 1KB fragment in 2 from 5 share policy

| S/N | FileSize | KeyShaCreTime | KeyShaWriTime | ShaRecTime | SecRecTime | OverHeadCost |
|---|---|---|---|---|---|---|
| 1 | 1KB | 0 | 0.007813275 | 0.001562595 | 0.007751107 | 0.017126977 |
| 2 | 10KB | 0.04972036 | 0.05682285 | 0.01562595 | 0 | 0.12216916 |
| 3 | 100KB | 0.3956114 | 0.6448381 | 0.008513689 | 0 | 1.048963189 |
| 4 | 1MB | 4.817735 | 7.716554 | 0.067607008 | 0.004066491 | 12.6059625 |
| 5 | 10MB | 106.03329 | 224.536643 | 0.748716545 | 0.365789056 | 331.6844386 |
| 6 | 100MB | 1538.0304 | 2280.3814 | 17.59926614 | 3.519875407 | 3839.530942 |

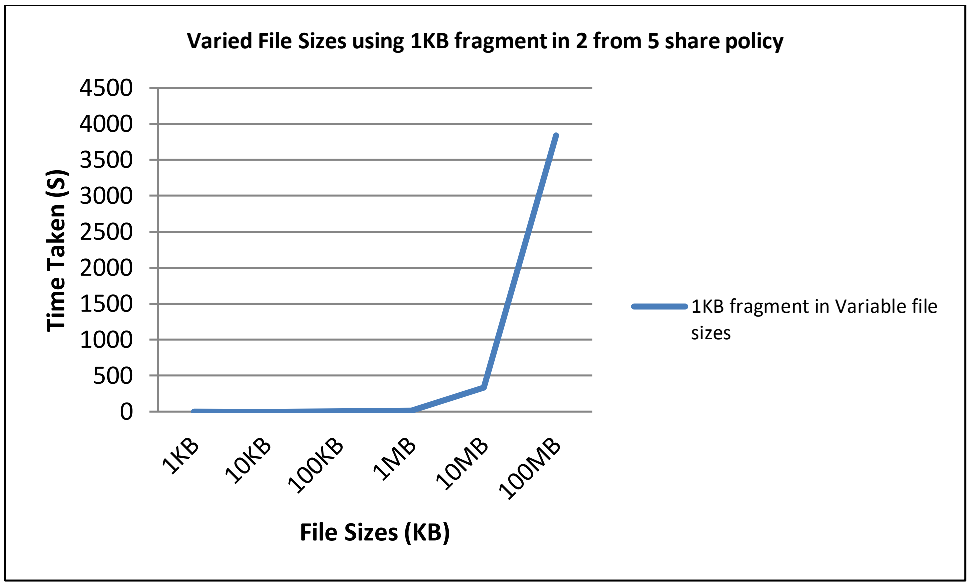


Figure 59: Key Share Creation and Recovering using 1KB fragment in 2 from 5 share policy

Table 40: Key Share Creation and Recovering using 10KB fragment in 2 from 5 share policy

| S/N | FileSize | KeyShaCreTime | KeyShaWriTime | ShaRecTime | SecRecTime | OverHeadCost |
|---|---|---|---|---|---|---|
| 1 | 1KB | 0.015624046 | 0.023435116 | 0.004688001 | 0.007812977 | 0.05156014 |
| 2 | 10KB | 0.019611657 | 0.015583277 | 0.003124714 | 0.00781095 | 0.046130598 |
| 3 | 100KB | 0.07102507 | 0.05682039 | 0.001562595 | 0 | 0.129408055 |
| 4 | 1MB | 0.3954959 | 0.6173189 | 0.010937715 | 0.0078125 | 1.031565015 |
| 5 | 10MB | 3.7395 | 5.776235 | 0.075003028 | 0.039065003 | 9.629803031 |
| 6 | 100MB | 38.96319 | 67.60683 | 0.702415705 | 0.343688607 | 107.6161243 |
| 7 | 1GB | 400.1159 | 779.7861 | 21.66762698 | 3.590544581 | 1205.160172 |

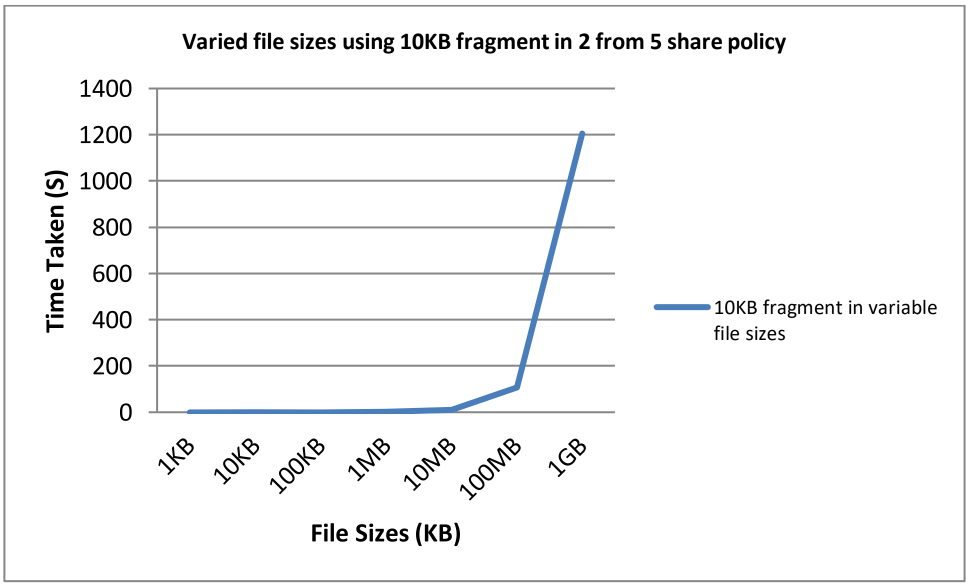


Figure 60: Key Share Creation and Recovering using 10KB fragment in 2 from 5 share policy

Table 41: Key Share Creation and Recovering using 100KB fragment in 2 from 5 share policy

| S/N | FileSize | KeyShaCreTime | KeyShaWriTime | ShaRecTime | SecRecTime | OverHeadCost |
|---|---|---|---|---|---|---|
| 1 | 1KB | 0.007838011 | 0.023381114 | 0.006269598 | 0 | 0.037488723 |
| 2 | 10KB | 0.007813931 | 0.015626431 | 0 | 0 | 0.023440362 |
| 3 | 100KB | 0.007812977 | 0.011719525 | 0.003098392 | 0.007735014 | 0.030365908 |
| 4 | 1MB | 0.09242752 | 0.12073322 | 0.003100586 | 0 | 0.216261326 |
| 5 | 10MB | 0.394453 | 0.6486295 | 0.006310487 | 0.015625954 | 1.065018941 |
| 6 | 100MB | 4.339445 | 6.764127 | 0.071877098 | 0.031312108 | 11.20676121 |
| 7 | 1GB | 42.49972 | 74.2406 | 0.722071505 | 0.453271508 | 117.915663 |

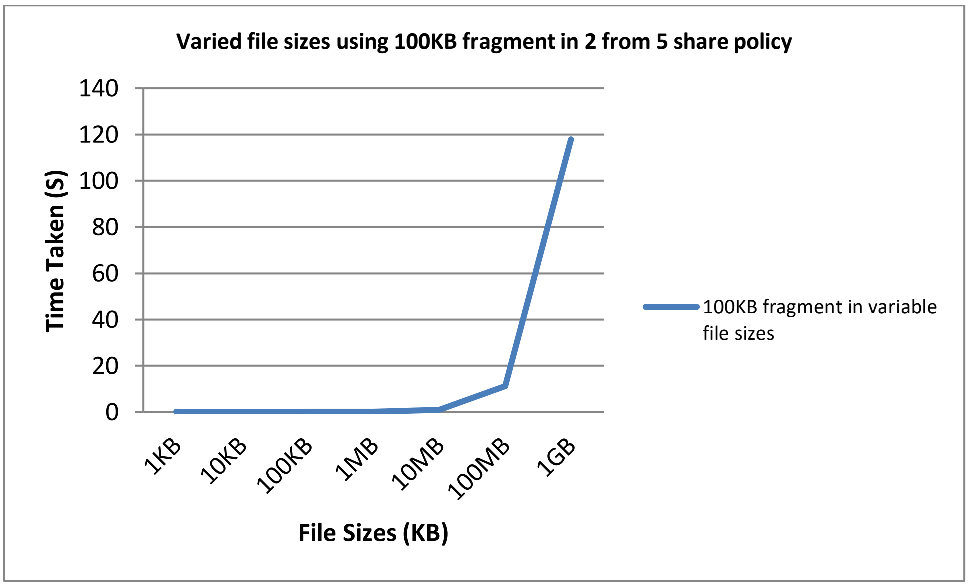


Figure 61: Key Share Creation and Recovering using 100KB fragment in 2 from 5 share policy

Table 42: Key Share Creation and Recovering using 1MB fragment in 2 from 5 share policy

| S/N | FileSize | KeyShaCreTime | KeyShaWriTime | ShaRecTime | SecRecTime | OverHeadCost |
|---|---|---|---|---|---|---|
| 1 | 1KB | 0.01562345 | 0.015625 | 0.001549006 | 0 | 0.032797456 |
| 2 | 10KB | 0 | 0.02343452 | 0.001101365 | 0.001501918 | 0.026037803 |
| 3 | 100KB | 0.011718452 | 0.050782264 | 0 | 0.003906727 | 0.066407443 |
| 4 | 1MB | 0.003876984 | 0.019532502 | 0 | 0 | 0.023409486 |
| 5 | 10MB | 0.08513852 | 0.09226236 | 0 | 0 | 0.17740088 |
| 6 | 100MB | 0.3869049 | 0.6269624 | 0.007811785 | 0 | 1.021679085 |
| 7 | 1GB | 4.906314 | 6.379165 | 0.084378815 | 0.078129053 | 11.44798687 |

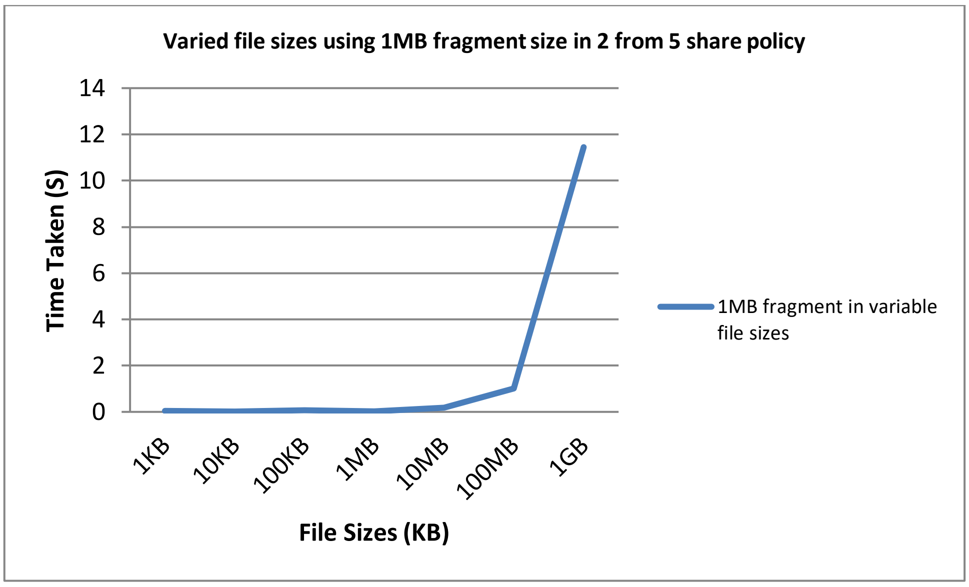


Figure 62: Key Share Creation and Recovering using 1MB fragment in 2 from 5 share policy

Table 43: Key Share Creation and Recovering using 10MB fragment in 2 from 5 share policy

| S/N | FileSize | KeyShaCreTime | KeyShaWriTime | ShaRecTime | SecRecTime | OverHeadCost |
|---|---|---|---|---|---|---|
| 1 | 1KB | 0.015615106 | 0.054621458 | 0.007812691 | 0 | 0.078049255 |
| 2 | 10KB | 0.015594959 | 0.031126022 | 0 | 0 | 0.046720982 |
| 3 | 100KB | 0.007812977 | 0.007812023 | 0.001579499 | 0 | 0.017204499 |
| 4 | 1MB | 0.007812977 | 0.015625596 | 0.00469821 | 0 | 0.028136783 |
| 5 | 10MB | 0 | 0.97619534 | 0.003150225 | 0 | 0.979345565 |
| 6 | 100MB | 0.04982027 | 0.07785375 | 0.003112006 | 0.007812977 | 0.138599003 |
| 7 | 1GB | 0.88994782 | 1.7235888 | 0.024221396 | 0.007811546 | 2.645569562 |

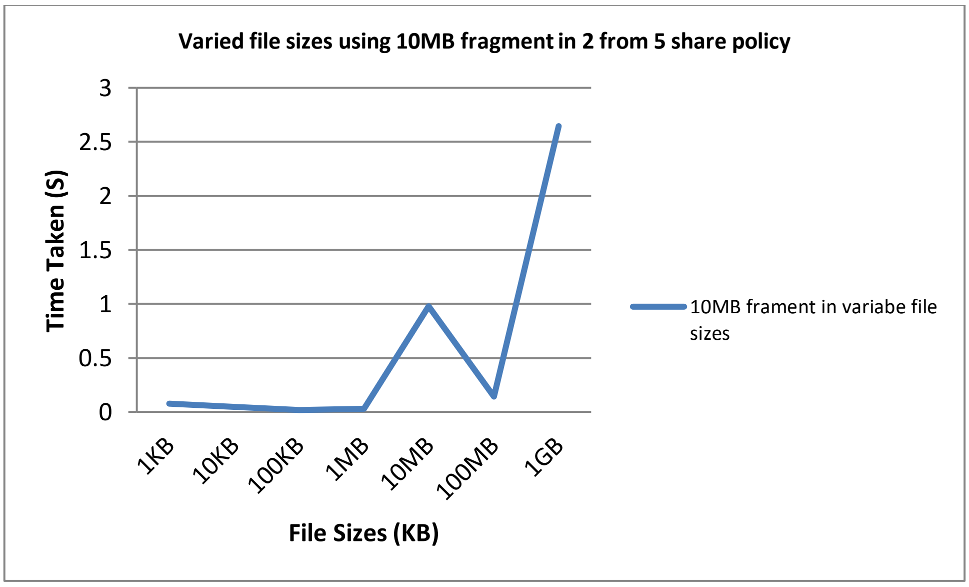


Figure 63: Key Share Creation and Recovering using 10MB fragment in 2 from 5 share policy

Table 44: Key Share Creation and Recovering using 100MB fragment in 2 from 5 share policy

| S/N | FileSize | KeyShaCreTime | KeyShaWriTime | ShaRecTime | SecRecTime | OverHeadCost |
|---|---|---|---|---|---|---|
| 1 | 1KB | 0.0079 | 0.007726431 | 0.00156281 | 0 | 0.017189241 |
| 2 | 10KB | 0 | 0.0078125 | 0.001549411 | 0 | 0.009361911 |
| 3 | 100KB | 0.007815003 | 0.007812977 | 0.00156419 | 0 | 0.01719217 |
| 4 | 1MB | 0.015667081 | 0.015568376 | 0 | 0 | 0.031235457 |
| 5 | 10MB | 0 | 0.007812977 | 0 | 0.007813096 | 0.015626073 |
| 6 | 100MB | 0.003906787 | 0.015624762 | 0.001571512 | 0 | 0.021103061 |
| 7 | 1GB | 0.19159794 | 0.16335043 | 0.00794549 | 0 | 0.36289386 |

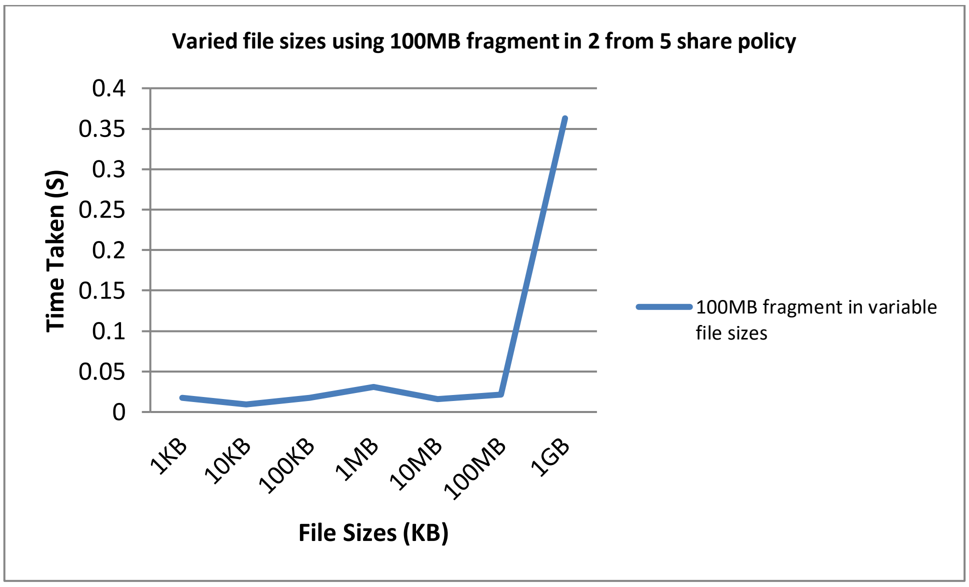


Figure 64: Key Share Creation and Recovering using 100MB fragment in 2 from 5 share policy

Table 45: Key Share Creation and Recovering using 1GB fragment in 2 from 5 share policy

| S/N | FileSize | KeyShaCreTime | KeyShaWriTime | ShaRecTime | SecRecTime | OverHeadCost |
|---|---|---|---|---|---|---|
| 1 | 1KB | 0.007737517 | 0.055197954 | 0 | 0 | 0.062935471 |
| 2 | 10KB | 0 | 0.015625 | 0.001582384 | 0 | 0.017207384 |
| 3 | 100KB | 0.009815454 | 0.011816621 | 0.002163911 | 0.000500083 | 0.024296069 |
| 4 | 1MB | 0.003503442 | 0.013517618 | 0.00800322 | 0.001502514 | 0.026526794 |
| 5 | 10MB | 0.007814407 | 0.00781405 | 0.001578999 | 0 | 0.017207456 |
| 6 | 100MB | 0.007820487 | 0.046877384 | 0.001562715 | 0.007751465 | 0.064012051 |
| 7 | 1GB | 1.119861722 | 0.605621219 | 0.012093687 | 0.007828951 | 1.745405579 |

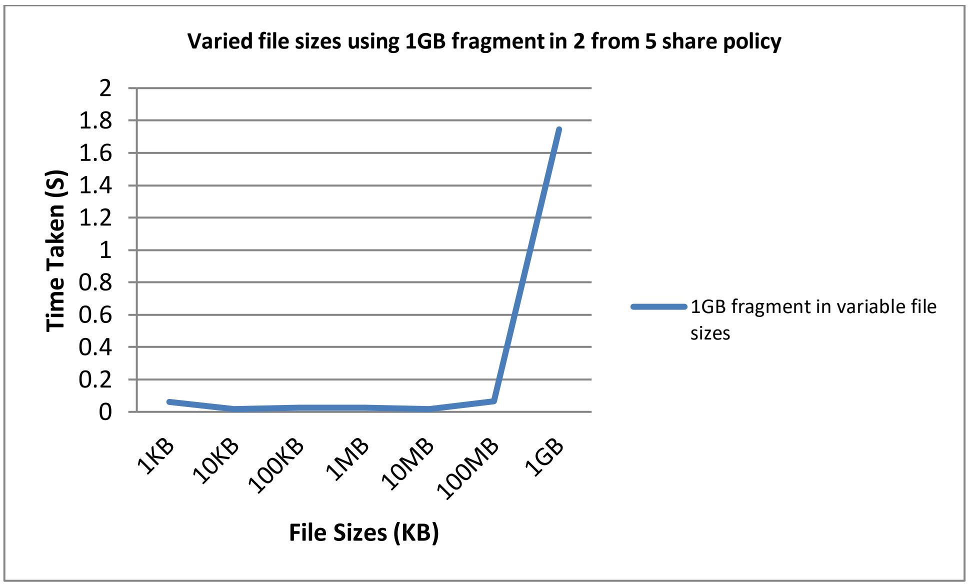


Figure 65: Key Share Creation and Recovering using 1GB fragment in 2 from 5 share policy

**File Sizes: 1KB, 10KB, 100KB, 1MB, 10MB, 100MB, 1GB**

**Fragment Sizes: 1KB, 10KB, 100KB, 1MB, 10MB, 100MB, 1GB**

**Secret Sharing Policy: 3 from 5**

**Plot: File Sizes in KB against Time Taken in Seconds to process and recover secret key using a particular key share policy.**

Table 46: Key Share Creation and Recovering using 1KB fragment in 3 from 5 share policy

| S/N | FileSize | KeyShaCreTime | KeyShaWriTime | ShaRecTime | SecRecTime | OverHeadCost |
|---|---|---|---|---|---|---|
| 1 | 1KB | 0.001999975 | 0.014819742 | 0.001568794 | 0 | 0.018388511 |
| 2 | 10KB | 0.04971905 | 0.07812608 | 0.002437305 | 0 | 0.130282435 |
| 3 | 100KB | 0.5164762 | 0.6092557 | 0.015612602 | 0.424601555 | 1.565946057 |
| 4 | 1MB | 5.293963 | 6.223017 | 0.069243789 | 41.463992 | 53.05021579 |
| 5 | 10MB | 47.87565 | 71.48502 | 0.693807769 | 4101.797584 | 4221.852062 |

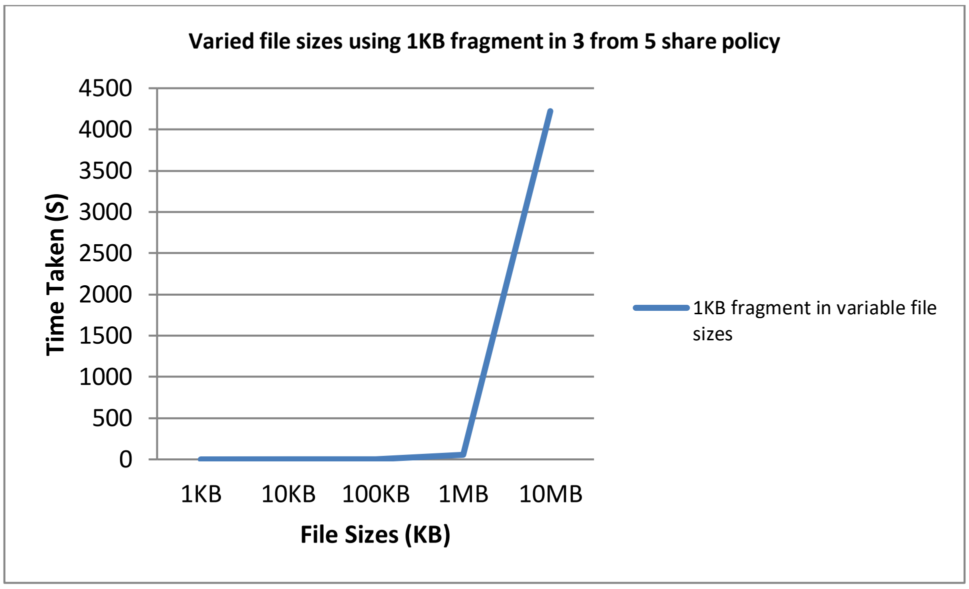


Figure 66: Key Share Creation and Recovering using 1KB fragment in 3 from 5 share policy

Table 47: Key Share Creation and Recovering using 10KB fragment in 3 from 5 share policy

| S/N | FileSize | KeyShaCreTime | KeyShaWriTime | ShaRecTime | SecRecTime | OverHeadCost |
|---|---|---|---|---|---|---|
| 1 | 1KB | 0.007918477 | 0.001565707 | 0.034364295 | 0 | 0.043848479 |
| 2 | 10KB | 0.011720224 | 0.01171881 | 0.002063322 | 0 | 0.025502356 |
| 3 | 100KB | 0.07813096 | 0.0781237 | 0.001562691 | 0.0078125 | 0.165629851 |
| 4 | 1MB | 0.6922069 | 0.6770385 | 0.007134318 | 0.424463511 | 1.800843229 |
| 5 | 10MB | 5.820417 | 6.456415 | 0.1079391 | 41.2689935 | 53.6537646 |
| 6 | 100MB | 49.52936 | 280.75698 | 0.698485422 | 4101.788009 | 4432.772834 |

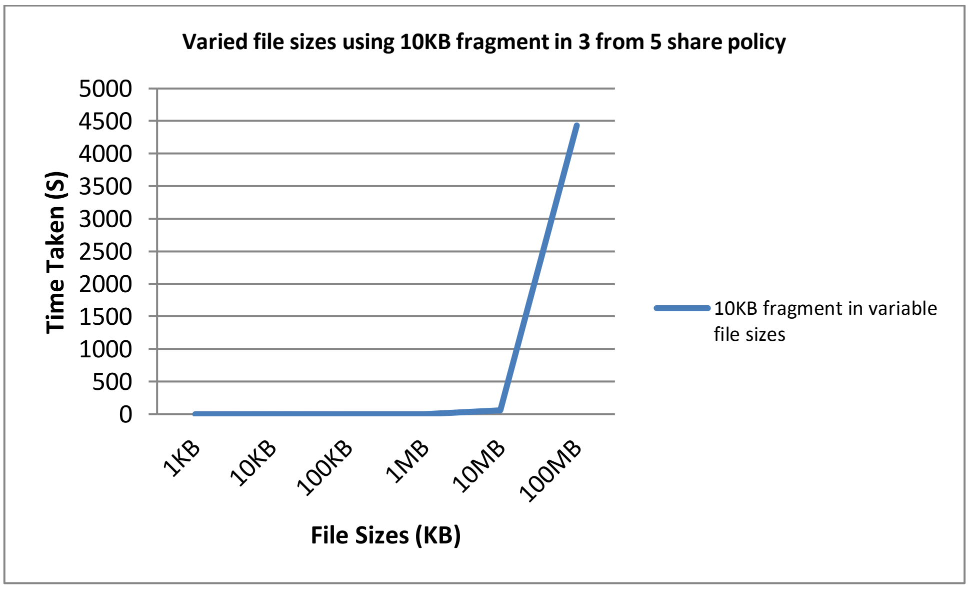


Figure 67: Key Share Creation and Recovering using 10KB fragment in 3 from 5 share policy

Table 48: Key Share Creation and Recovering using 100KB fragment in 3 from 5 share policy

| S/N | FileSize | KeyShaCreTime | KeyShaWriTime | ShaRecTime | SecRecTime | OverHeadCost |
|---|---|---|---|---|---|---|
| 1 | 1KB | 0.014020085 | 0.038558364 | 0.006610155 | 0.005006909 | 0.064195513 |
| 2 | 10KB | 0.010513544 | 0.031795979 | 0.0038059 | 0.003003597 | 0.04911902 |
| 3 | 100KB | 0.004733075 | 0.018936558 | 0.005607772 | 0.00801003 | 0.037287435 |
| 4 | 1MB | 0.04486291 | 0.17686271 | 0.011015558 | 0.424120426 | 0.656861604 |
| 5 | 10MB | 0.4684845 | 1.8965906 | 0.075609422 | 41.84533298 | 44.2860175 |
| 6 | 100MB | 4.66078 | 19.933848 | 0.90046041 | 4531.828259 | 4557.323347 |

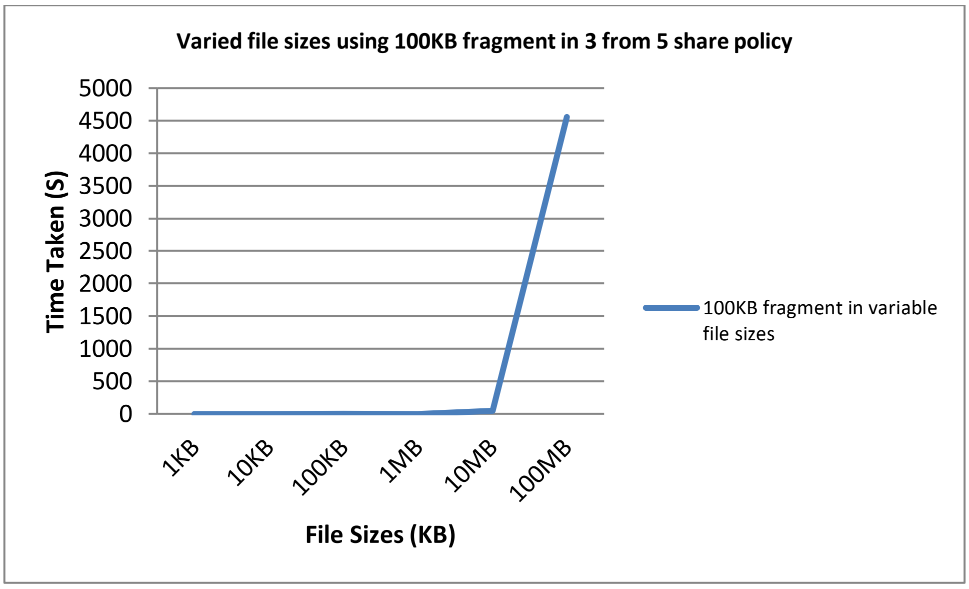


Figure 68: Key Share Creation and Recovering using 100KB fragment in 3 from 5 share policy

Table 49: Key Share Creation and Recovering using 1MB fragment in 3 from 5 share policy

| S/N | FileSize | KeyShaCreTime | KeyShaWriTime | ShaRecTime | SecRecTime | OverHeadCost |
|---|---|---|---|---|---|---|
| 1 | 1KB | 0.00750792 | 0.024041891 | 0.003505302 | 0.003004074 | 0.038059187 |
| 2 | 10KB | 0.005506635 | 0.024134517 | 0.003405094 | 0.002502441 | 0.035548687 |
| 3 | 100KB | 0.005005479 | 0.024034381 | 0.003807426 | 0.00250411 | 0.035351396 |
| 4 | 1MB | 0.005508065 | 0.019026518 | 0.003905606 | 0.003003359 | 0.031443548 |
| 5 | 10MB | 0.04505467 | 0.18677616 | 0.004206061 | 0.018526435 | 0.254563326 |
| 6 | 100MB | 0.4826415 | 1.946952 | 0.011116862 | 0.432633519 | 2.873343881 |
| 7 | 1GB | 4.989744 | 21.453339 | 0.187532067 | 41.61609554 | 68.24671061 |

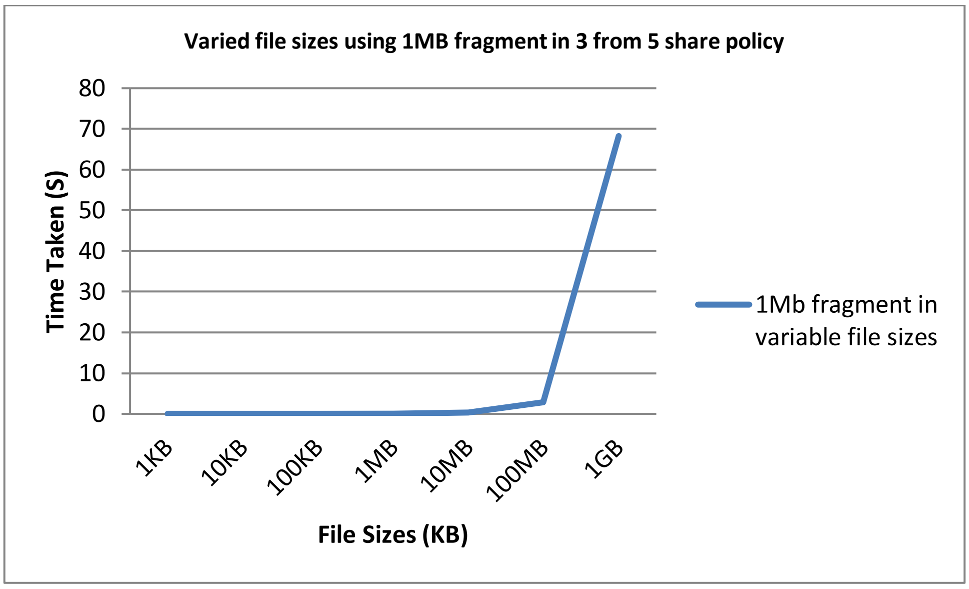


Figure 69: Key Share Creation and Recovering using 1MB fragment in 3 from 5 share policy

Table 50: Key Share Creation and Recovering using 10MB fragment in 3 from 5 share policy

| S/N | FileSize | KeyShaCreTime | KeyShaWriTime | ShaRecTime | SecRecTime | OverHeadCost |
|---|---|---|---|---|---|---|
| 1 | 1KB | 0.006509542 | 0.030044436 | 0.004006076 | 0.003504992 | 0.044065046 |
| 2 | 10KB | 0.005507469 | 0.019529939 | 0.003304434 | 0.002000451 | 0.030342293 |
| 3 | 100KB | 0.004006028 | 0.020030022 | 0.00320487 | 0.002504587 | 0.029745507 |
| 4 | 1MB | 0.005007386 | 0.023534417 | 0.00390588 | 0.002505541 | 0.034953224 |
| 5 | 10MB | 0.004507065 | 0.03955853 | 0.003304601 | 0.002004027 | 0.049374223 |
| 6 | 100MB | 0.05157232 | 0.18577373 | 0.004005647 | 0.006009936 | 0.247361633 |
| 7 | 1GB | 1.6381979 | 5.3205009 | 0.17423861 | 0.533988953 | 7.666926363 |

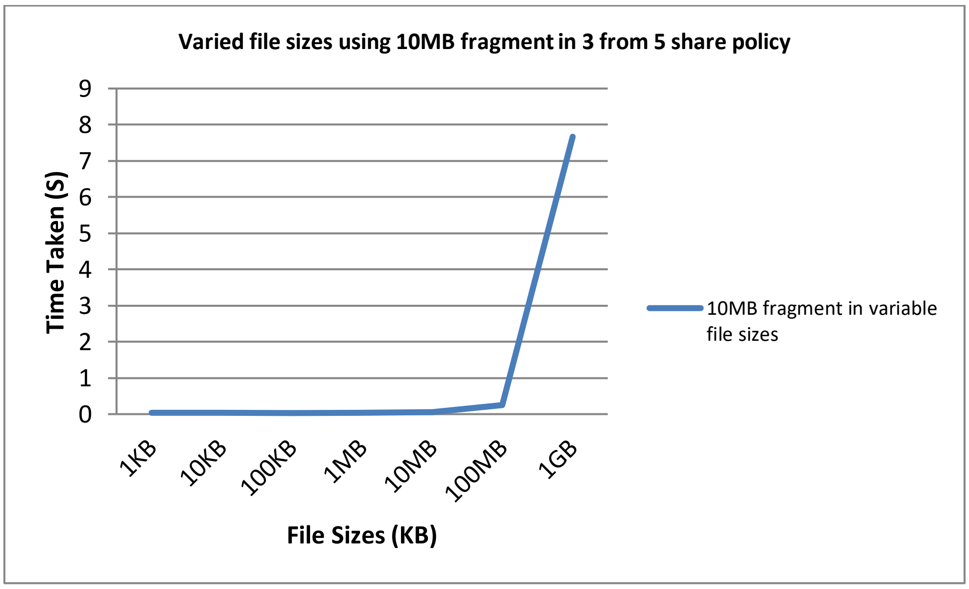


Figure 70: Key Share Creation and Recovering using 10MB fragment in 3 from 5 share policy

Table 51: Key Share Creation and Recovering using 100MB fragment in 3 from 5 share policy

| S/N | FileSize | KeyShaCreTime | KeyShaWriTime | ShaRecTime | SecRecTime | OverHeadCost |
|---|---|---|---|---|---|---|
| 1 | 1KB | 0.009513378 | 0.028017998 | 0.00420537 | 0.003503919 | 0.045240665 |
| 2 | 10KB | 0.013522625 | 0.046231985 | 0.00730865 | 0.00600493 | 0.07306819 |
| 3 | 100KB | 0.01452148 | 0.029041052 | 0.005407834 | 0.004507065 | 0.053477431 |
| 4 | 1MB | 0.005006552 | 0.018527627 | 0.00300386 | 0.002505064 | 0.029043103 |
| 5 | 10MB | 0.005009532 | 0.025157452 | 0.004906583 | 0.003504038 | 0.038577605 |
| 6 | 100MB | 0.019026995 | 0.035051942 | 0.003906035 | 0.002503037 | 0.060488009 |
| 7 | 1GB | 0.58460409 | 0.70191026 | 0.036320114 | 0.013153458 | 1.335987922 |

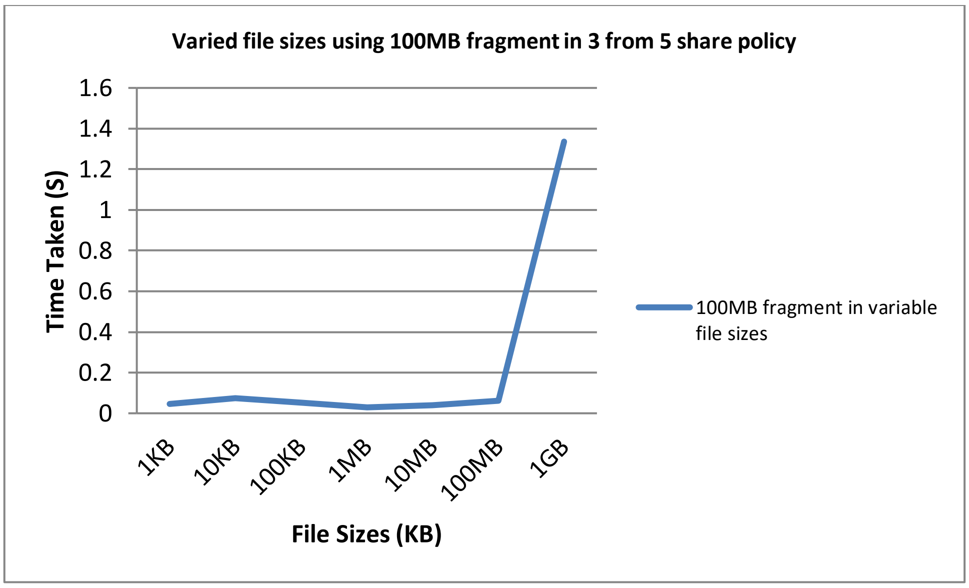


Figure 71: Key Share and Recovering using 100MB fragment in 3 from 5 share policy

Table 52: Key Share Creation and Recovering using 1GB fragment in 3 from 5 share policy

| S/N | FileSize | KeyShaCreTime | KeyShaWriTime | ShaRecTime | SecRecTime | OverHeadCost |
|---|---|---|---|---|---|---|
| 1 | 1KB | 0.004504561 | 0.022533417 | 0.003804874 | 0.002502918 | 0.03334577 |
| 2 | 10KB | 0.004504919 | 0.02116394 | 0.003404713 | 0.0028162 | 0.031889772 |
| 3 | 100KB | 0.004005432 | 0.020029902 | 0.003505039 | 0.002503037 | 0.03004341 |
| 4 | 1MB | 0.009513497 | 0.034051538 | 0.005709576 | 0.002499104 | 0.051773715 |
| 5 | 10MB | 0.005006075 | 0.031042457 | 0.00410662 | 0.002002001 | 0.042157153 |
| 6 | 100MB | 0.01941359 | 0.02503264 | 0.004306817 | 0.003006458 | 0.051759505 |
| 7 | 1GB | 3.183449984 | 1.58512044 | 0.382245779 | 0.337004185 | 5.487820388 |

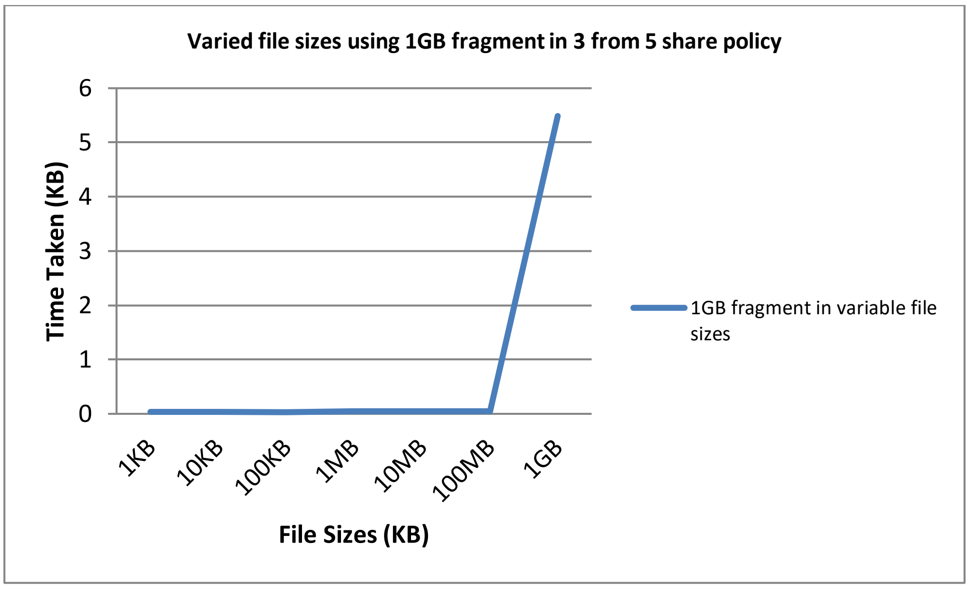


Figure 72: Key Share Creation and Recovering using 1GB fragment in 3 from 5 share policy

**File Sizes: 1KB, 10KB, 100KB, 1MB, 10MB, 100MB, 1GB**

**Fragment Sizes: 1KB, 10KB, 100KB, 1MB, 10MB, 100MB, 1GB**

**Secret Sharing Policy: 4 from 5**

**Plot: File Sizes in KB against Time Taken in Seconds to process and recover secret key using a particular key share policy.**

Table 53: Key Share Creation and Recovering using 1KB fragment in 4 from 5 share policy

| S/N | FileSize | KeyShaCreTime | KeyShaWriTime | ShaRecTime | SecRecTime | OverHeadCost |
|---|---|---|---|---|---|---|
| 1 | 1KB | 0.005007386 | 0.025519967 | 0.00380497 | 0.005005836 | 0.039338159 |
| 2 | 10KB | 0.05279541 | 0.19847545 | 0.00460763 | 0.123180866 | 0.379059356 |
| 3 | 100KB | 0.5393359 | 2.00651 | 0.011215639 | 96.81914401 | 99.37620555 |

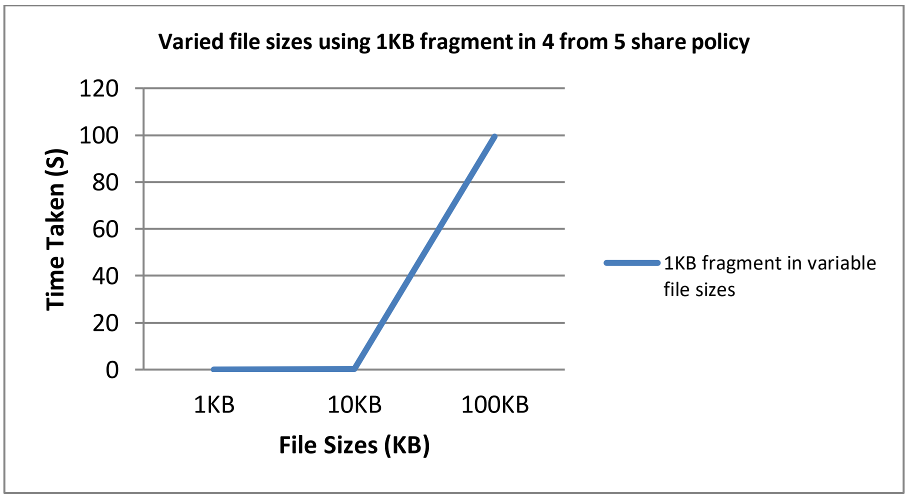


Figure 73: Key Share Creation and Recovering using 1KB fragment in 4 from 5 share policy

Table 54: Key Share Creation and Recovering using 10KB fragment in 4 from 5 share policy

| S/N | FileSize | KeyShaCreTime | KeyShaWriTime | ShaRecTime | SecRecTime | OverHeadCost |
|---|---|---|---|---|---|---|
| 1 | 1KB | 0.007010937 | 0.022032022 | 0.003404284 | 0.004004002 | 0.036451245 |
| 2 | 10KB | 0.00500536 | 0.020031929 | 0.003805208 | 0.005007982 | 0.033850479 |
| 3 | 100KB | 0.05189267 | 0.20212195 | 0.004207563 | 0.116168022 | 0.374390205 |
| 4 | 1MB | 0.5541734 | 1.9436804 | 0.011416817 | 101.3007209 | 103.8099915 |

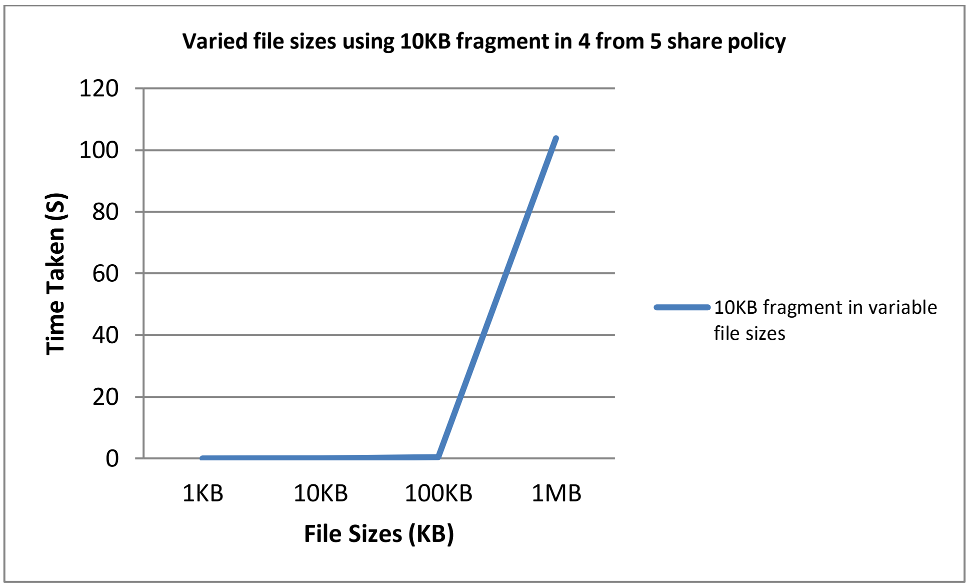


Figure 74: Key Share Creation and Recovering using 10KB fragment in 4 from 5 share policy

Table 55: Key Share Creation and Recovering using 100KB fragment in 4 from 5 share policy

| S/N | FileSize | KeyShaCreTime | KeyShaWriTime | ShaRecTime | SecRecTime | OverHeadCost |
|---|---|---|---|---|---|---|
| 1 | 1KB | 0.00600791 | 0.028062105 | 0.003203535 | 0.004007101 | 0.041280651 |
| 2 | 10KB | 0.005008936 | 0.025034904 | 0.004005957 | 0.004005909 | 0.038055706 |
| 3 | 100KB | 0.006007075 | 0.020029426 | 0.003605556 | 0.005007029 | 0.034649086 |
| 4 | 1MB | 0.05279888 | 0.2021174 | 0.004406166 | 0.115168095 | 0.374490541 |
| 5 | 10MB | 0.5434386 | 2.2062003 | 0.011215019 | 102.727818 | 105.4886719 |

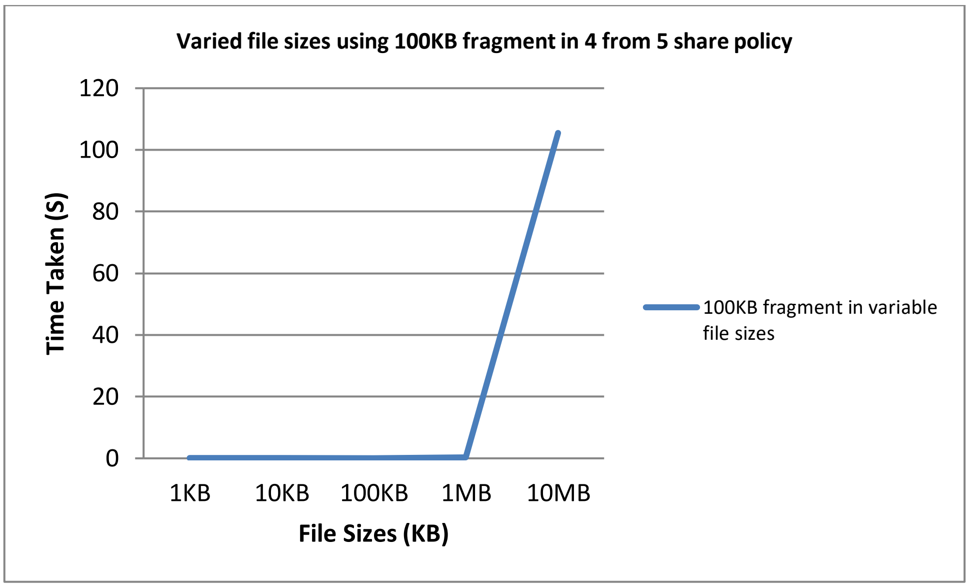


Figure 75: Key Share Creation and Recovering using 100KB fragment in 4 from 5 share policy

Table 56: Key Share Creation and Recovering using 1MB fragment in 4 from 5 share policy

| S/N | FileSize | KeyShaCreTime | KeyShaWriTime | ShaRecTime | SecRecTime | OverHeadCost |
|---|---|---|---|---|---|---|
| 1 | 1KB | 0.005009174 | 0.019028902 | 0.003806543 | 0.004006147 | 0.031850767 |
| 2 | 10KB | 0.005007982 | 0.023032904 | 0.004005384 | 0.004005909 | 0.036052179 |
| 3 | 100KB | 0.006009102 | 0.039059877 | 0.004007149 | 0.003004074 | 0.052080202 |
| 4 | 1MB | 0.020028472 | 0.041079044 | 0.012418222 | 0.009011984 | 0.082537722 |
| 5 | 10MB | 0.06737124 | 0.21885027 | 0.004606152 | 0.116168976 | 0.406996638 |
| 6 | 100MB | 0.6781625 | 2.3552517 | 0.011617374 | 97.11757994 | 100.1626115 |

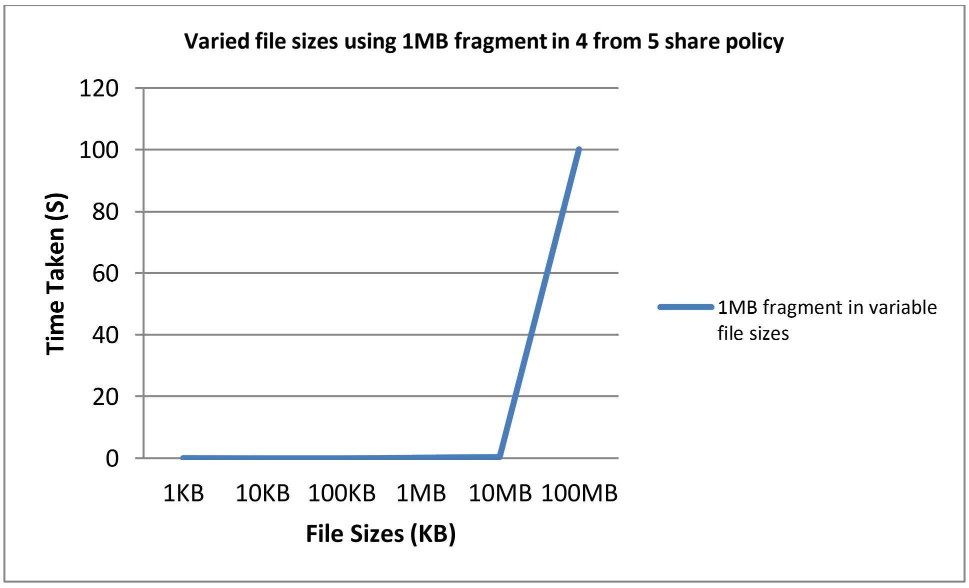


Figure 76: Key Share Creation and Recovering using 1MB fragment in 4 from 5 share policy

Table 57: Key Share Creation and Recovering using 10MB fragment in 4 from 5 share policy

| S/N | FileSize | KeyShaCreTime | KeyShaWriTime | ShaRecTime | SecRecTime | OverHeadCost |
|---|---|---|---|---|---|---|
| 1 | 1KB | 0.017022848 | 0.036051989 | 0.00580821 | 0.007007122 | 0.065890169 |
| 2 | 10KB | 0.00600791 | 0.022032976 | 0.003603983 | 0.004000902 | 0.035645771 |
| 3 | 100KB | 0.010014057 | 0.041062117 | 0.0038064 | 0.004008055 | 0.058890629 |
| 4 | 1MB | 0.007007122 | 0.028040886 | 0.004407072 | 0.005008936 | 0.044464016 |
| 5 | 10MB | 0.005006433 | 0.018527508 | 0.003606033 | 0.004004955 | 0.031144929 |
| 6 | 100MB | 0.06571531 | 0.25674928 | 0.004205561 | 0.114168167 | 0.440838318 |
| 7 | 1GB | 2.4023419 | 6.7281839 | 0.22472744 | 104.130311 | 113.4855643 |

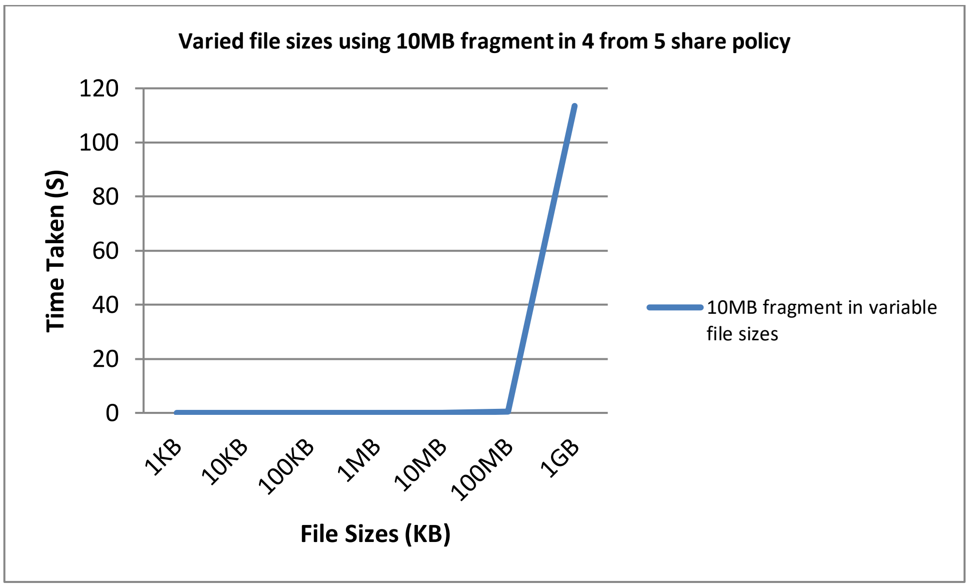


Figure 77: Key Share Creation and Recovering using 10MB fragment in 4 from 5 share policy

Table 58: Key Share Creation and Recovering using 100MB fragment in 4 from 5 share policy

| S/N | FileSize | KeyShaCreTime | KeyShaWriTime | ShaRecTime | SecRecTime | OverHeadCost |
|---|---|---|---|---|---|---|
| 1 | 1KB | 0.029040098 | 0.052077055 | 0.015822411 | 0.010015965 | 0.106955529 |
| 2 | 10KB | 0.005007029 | 0.022032976 | 0.003605461 | 0.004004955 | 0.034650421 |
| 3 | 100KB | 0.016027927 | 0.034050941 | 0.006007957 | 0.007007122 | 0.063093948 |
| 4 | 1MB | 0.011016846 | 0.034053087 | 0.00600996 | 0.007009983 | 0.058089876 |
| 5 | 10MB | 0.006008148 | 0.021037102 | 0.003604174 | 0.004004002 | 0.034653426 |
| 6 | 100MB | 0.010514975 | 0.028539062 | 0.015069962 | 0.02402997 | 0.078153969 |
| 7 | 1GB | 0.35621665 | 0.50816536 | 0.023654175 | 0.209298849 | 1.097335034 |

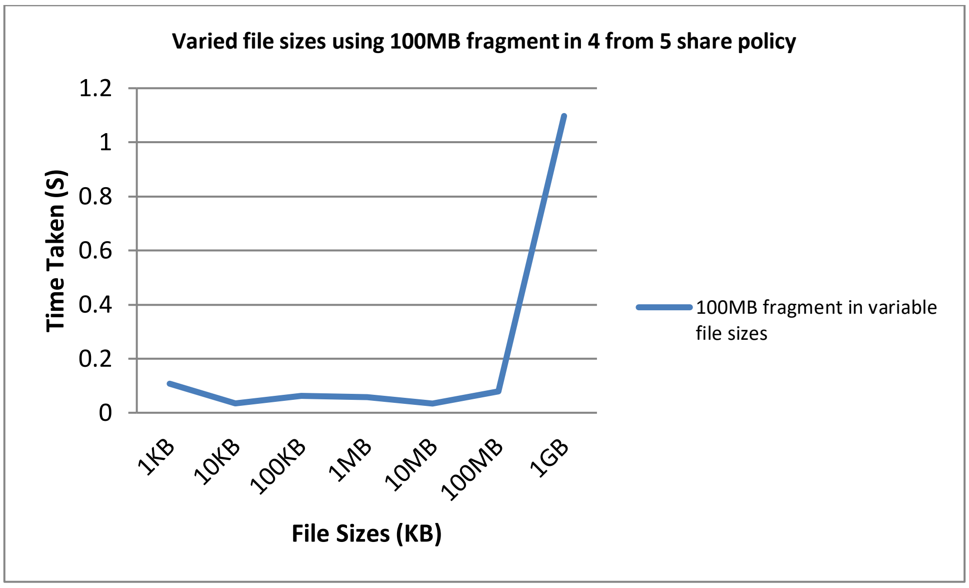


Figure 78: Key Share Creation and Recovering using 100MB fragment in 4 from 5 share policy

Table 59: Key Share Creation and Recovering using 1GB fragment in 4 from 5 share policy

| S/N | FileSize | KeyShaCreTime | KeyShaWriTime | ShaRecTime | SecRecTime | OverHeadCost |
|---|---|---|---|---|---|---|
| 1 | 1KB | 0.026038885 | 0.095140934 | 0.013951063 | 0.08305788 | 0.218188763 |
| 2 | 10KB | 0.010014057 | 0.030047894 | 0.005007219 | 0.006008863 | 0.051078033 |
| 3 | 100KB | 0.021031857 | 0.063093185 | 0.006408787 | 0.00801301 | 0.098546839 |
| 4 | 1MB | 0.011013985 | 0.033047199 | 0.006009436 | 0.007011175 | 0.057081795 |
| 5 | 10MB | 0.005006075 | 0.020029068 | 0.003403807 | 0.003004074 | 0.031443024 |
| 6 | 100MB | 0.027036905 | 0.036056995 | 0.003605795 | 0.004007101 | 0.070706797 |
| 7 | 1GB | 1.863462448 | 1.334843516 | 0.088929844 | 0.956404209 | 4.243640017 |

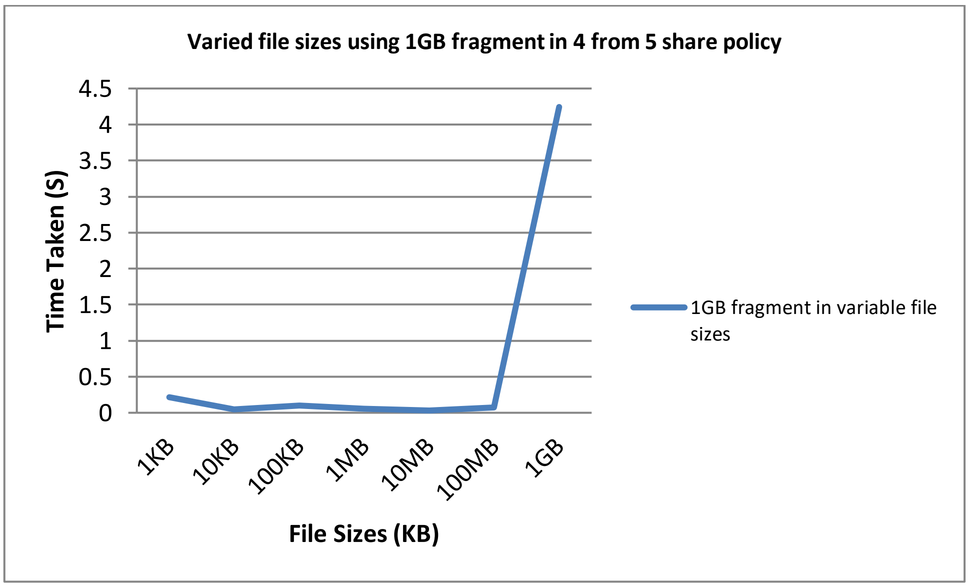


Figure 79: Key Share Creation and Recovering using 1GB fragment in 4 from 5 share policy

## 12.3 Variant Two

### 12.3.1 Fragments

**File Sizes: 1KB, 10KB, 100KB, 1MB, 10MB, 100MB, 1GB**

**Fragment Sizes: 15% of each file Size**

**Secret Sharing Policy: 2 from 5**

**Plot: File Sizes in KB against Time Taken in Seconds to process and recombine file using varied key share policies.**

Table 60: Varied file sizes using equal number of fragments in 2 from 5 share policy

| S/N | FileSize | FileSplitTime | FragEncTime | FragDecTime | FileComTime | OverHeadCost |
|---|---|---|---|---|---|---|
| 1 | 1KB | 0.002925038 | 0.005387 | 0.013149977 | 0.001445055 | 0.02290707 |
| 2 | 10KB | 0.038562179 | 0.0073225 | 0.03125155 | 0.002905488 | 0.080041716 |
| 3 | 100KB | 0.003394961 | 0.0068255 | 0.035158157 | 0.002437949 | 0.047816568 |
| 4 | 1MB | 0.004876733 | 0.009764 | 0.020504594 | 0.002928138 | 0.038073464 |
| 5 | 10MB | 0.022940755 | 0.0542355 | 0.08544898 | 0.01366663 | 0.176291865 |
| 6 | 100MB | 0.263669968 | 0.7352945 | 0.684566617 | 0.190426588 | 1.873957673 |
| 7 | 1GB | 26.31982481 | 18.0551 | 19.57863176 | 14.14794874 | 78.10150532 |

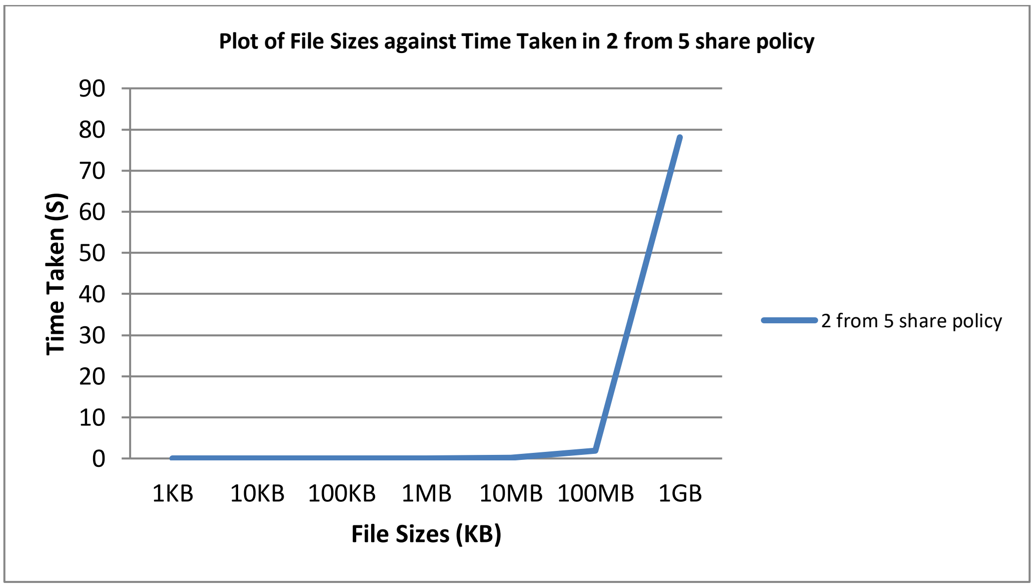


Figure 80: Varied file sizes using equal number of fragments in 2 from 5 share policy

Table 61: Varied file sizes using equal number of fragments in 3 from 5 share policy

| S/N | FileSize | FileSplitTime | FragEncTime | FragDecTime | FileComTime | OverHeadCost |
|---|---|---|---|---|---|---|
| 1 | 1KB | 0.01074183 | 0.007808 | 0.0117203 | 0.001464128 | 0.031734258 |
| 2 | 10KB | 0.003417015 | 0.005401 | 0.014142394 | 0.001951933 | 0.024912342 |
| 3 | 100KB | 0.003919721 | 0.0097795 | 0.013170838 | 0.001954079 | 0.028824138 |
| 4 | 1MB | 0.006330132 | 0.0097585 | 0.016113043 | 0.003416777 | 0.035618452 |
| 5 | 10MB | 0.024902463 | 0.066904 | 0.079588175 | 0.016105175 | 0.187499813 |
| 6 | 100MB | 0.249003887 | 0.499052 | 0.668944597 | 0.171869397 | 1.588869882 |
| 7 | 1GB | 29.3764677 | 18.272035 | 19.59326553 | 14.14258373 | 81.38435197 |

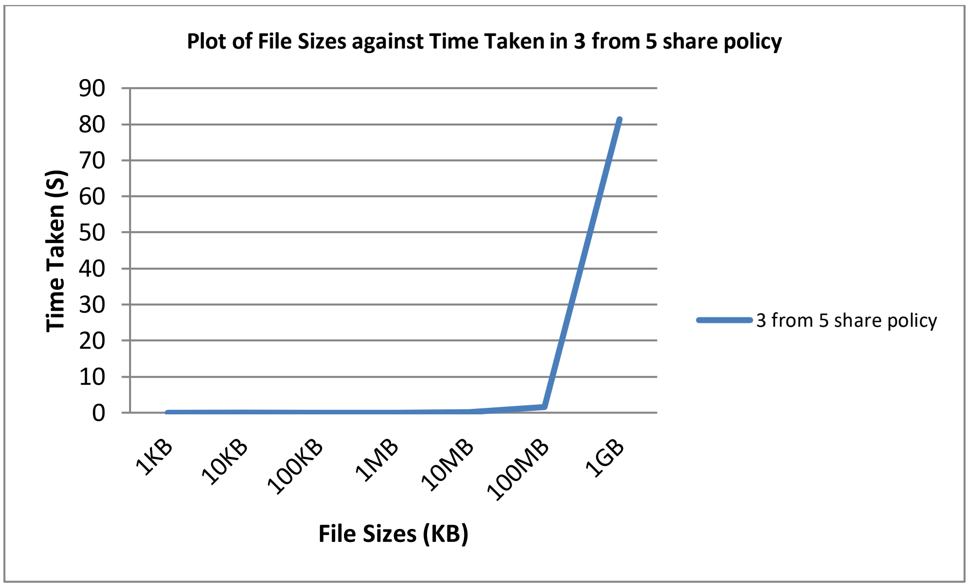


Figure 81: Varied file sizes using equal number of fragments in 3 from 5 share policy

Table 62: Varied file sizes using equal number of fragments in 4 from 5 share policy

| S/N | FileSize | FileSplitTime | FragEncTime | FragDecTime | FileComTime | OverHeadCost |
|---|---|---|---|---|---|---|
| 1 | 1KB | 0.002427101 | 0.0053805 | 0.00732398 | 0.00145793 | 0.016589511 |
| 2 | 10KB | 0.00292635 | 0.005372 | 0.008307219 | 0.001480222 | 0.01808579 |
| 3 | 100KB | 0.003412962 | 0.004893 | 0.012213349 | 0.001952171 | 0.022471483 |
| 4 | 1MB | 0.004395008 | 0.01222 | 0.014651418 | 0.002941012 | 0.034207438 |
| 5 | 10MB | 0.028311729 | 0.063499 | 0.06786871 | 0.013174653 | 0.172854092 |
| 6 | 100MB | 0.220714808 | 0.5121775 | 0.73241353 | 0.276359558 | 1.741665396 |
| 7 | 1GB | 26.46483827 | 18.09913 | 19.42675602 | 14.02051485 | 78.01123914 |

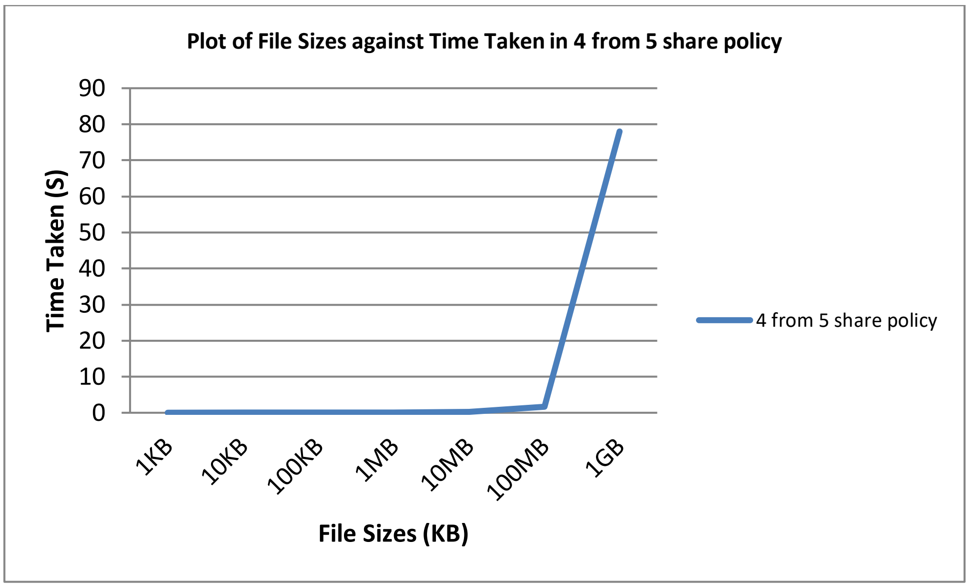


Figure 82: Varied file sizes using equal number of fragments in 4 from 5 share policy

Table 63: Varied file sizes using equal number of fragments in 4 from 10 share policy

| S/N | FileSize | FileSplitTime | FragEncTime | FragDecTime | FileComTime | OverHeadCost |
|---|---|---|---|---|---|---|
| 1 | 1KB | 0.026849508 | 0.00783 | 0.03028059 | 0.00194037 | 0.066900468 |
| 2 | 10KB | 0.004398942 | 0.0068895 | 0.024399281 | 0.002448678 | 0.038136401 |
| 3 | 100KB | 0.019526243 | 0.007301 | 0.030745983 | 0.001979351 | 0.059552577 |
| 4 | 1MB | 0.004876614 | 0.006857 | 0.041999578 | 0.003415108 | 0.0571483 |
| 5 | 10MB | 0.030759573 | 0.0800605 | 0.073757172 | 0.028308153 | 0.212885398 |
| 6 | 100MB | 0.262196779 | 0.581529 | 0.638176203 | 0.275382519 | 1.757284501 |
| 7 | 1GB | 35.07715082 | 18.72004 | 20.2573272 | 14.28020155 | 88.33471957 |

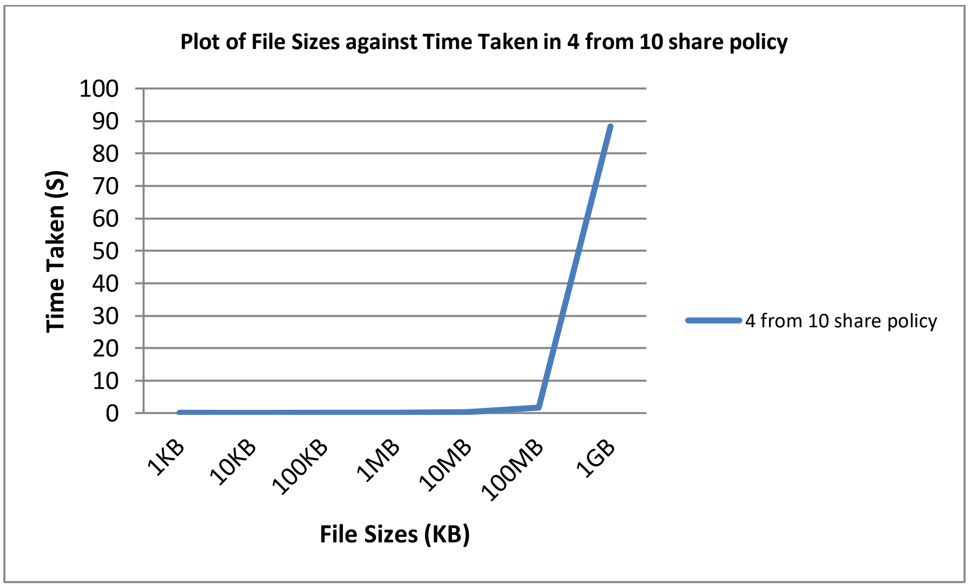


Figure 83: Varied file sizes using equal number of fragments in 4 from 10 share policy

Table 64: Varied file sizes using equal number of fragments in 6 from 10 share policy

| S/N | FileSize | FileSplitTime | FragEncTime | FragDecTime | FileComTime | OverHeadCost |
|---|---|---|---|---|---|---|
| 1 | 1KB | 0.002924442 | 0.005363 | 0.014597535 | 0.001461148 | 0.024346126 |
| 2 | 10KB | 0.002439857 | 0.003923 | 0.012702703 | 0.001458526 | 0.020524086 |
| 3 | 100KB | 0.00341785 | 0.0092475 | 0.012699008 | 0.001951814 | 0.027316171 |
| 4 | 1MB | 0.005369663 | 0.009251 | 0.01805234 | 0.008789301 | 0.041462304 |
| 5 | 10MB | 0.028310537 | 0.0502505 | 0.080553055 | 0.013691425 | 0.172805517 |
| 6 | 100MB | 0.231923342 | 0.479033 | 0.621574283 | 0.16698122 | 1.499511845 |
| 7 | 1GB | 27.49071908 | 18.178235 | 20.24266446 | 14.06738889 | 79.97900743 |

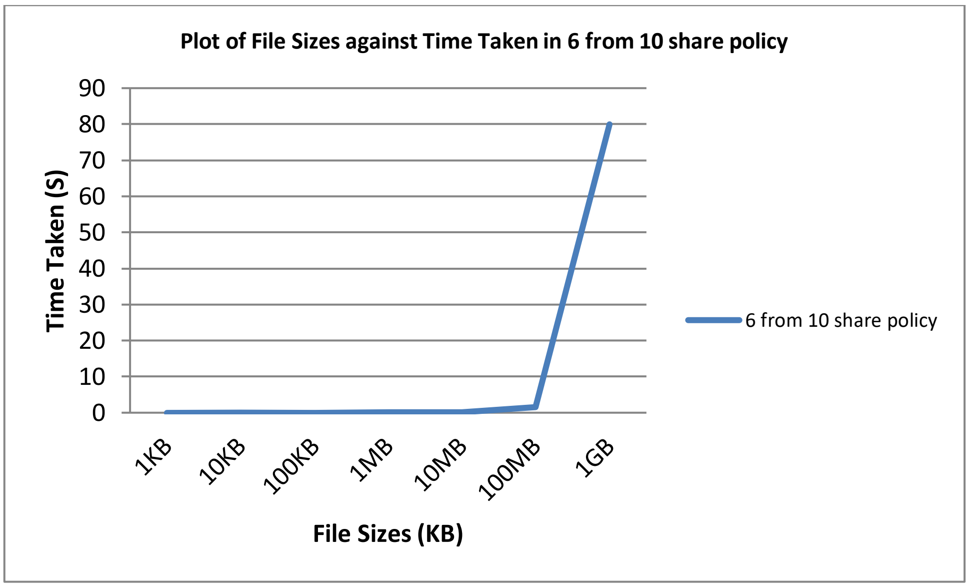


Figure 84: Varied file sizes using equal number of fragments in 6 from 10 share policy

Table 65: Varied file sizes using equal number of fragments in 8 from 10 share policy

| S/N | FileSize | FileSplitTime | FragEncTime | FragDecTime | FileComTime | OverHeadCost |
|---|---|---|---|---|---|---|
| 1 | 1KB | 0.019048691 | 0.0073495 | 0.024884462 | 0.001938701 | 0.053221354 |
| 2 | 10KB | 0.003939748 | 0.0083285 | 0.102038383 | 0.002929687 | 0.117236319 |
| 3 | 100KB | 0.002926469 | 0.005854 | 0.02783668 | 0.001965404 | 0.038582553 |
| 4 | 1MB | 0.004895687 | 0.009758 | 0.044403076 | 0.045897126 | 0.104953889 |
| 5 | 10MB | 0.035639167 | 0.0532065 | 0.154280305 | 0.018523335 | 0.261649307 |
| 6 | 100MB | 0.216794372 | 0.4418855 | 0.792963624 | 0.328602195 | 1.78024569 |
| 7 | 1GB | 37.22312522 | 19.845195 | 24.66403377 | 14.36472654 | 96.09708053 |

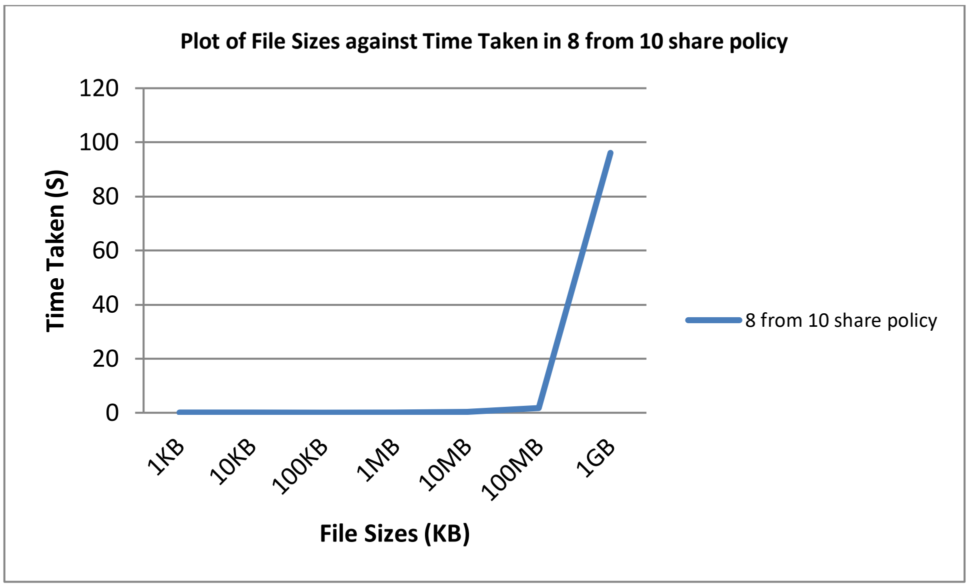


Figure 85: Varied file sizes using equal number of fragments in 8 from 10 share policy

### 12.3.2 Key Share creation and Recovering

**File Sizes: 1KB, 10KB, 100KB, 1MB, 10MB, 100MB, 1GB**

**Fragment Sizes: 15% of each file Size**

**Secret Sharing Policy: 2 from 5**

**Plot: File Sizes in KB against Time Taken in Seconds to process and recover secret key using varied key share policies.**

Table 66: Varied file sizes using equal number of fragments in 2 from 5 share policy

| S/N | FileSize | KeyShaCreTime | KeyShaWriTime | ShaRecTime | SecRecTime | OverHeadCost |
|---|---|---|---|---|---|---|
| 1 | 1KB | 0.0063395 | 7.08288 | 0.00391 | 0.001009345 | 7.094138845 |
| 2 | 10KB | 0.0063515 | 7.24591 | 0.0014255 | 0.000486374 | 7.254173374 |
| 3 | 100KB | 0.007299 | 7.3756 | 0.00196 | 0.000485539 | 7.385344539 |
| 4 | 1MB | 0.006354 | 7.66259 | 0.00098548 | 0.000488043 | 7.670417523 |
| 5 | 10MB | 0.007364 | 8.44544 | 0.001933122 | 0.00096786 | 8.455704982 |
| 6 | 100MB | 0.0073195 | 7.2813 | 0.00098455 | 0.00097096 | 7.290575009 |
| 7 | 1GB | 0.0083555 | 6.7577 | 0.001965 | 0 | 6.7680205 |

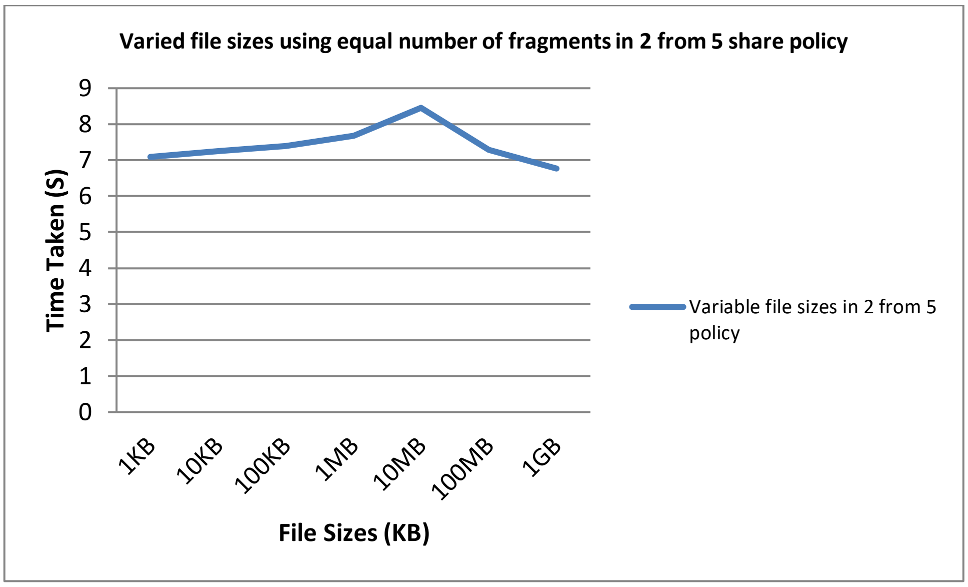


Figure 86: Varied file sizes using equal number of fragments in 2 from 5 share policy

Table 67: Varied file sizes using equal number of fragments in 3 from 5 share policy

| S/N | FileSize | KeyShaCreTime | KeyShaWriTime | ShaRecTime | SecRecTime | OverHeadCost |
|---|---|---|---|---|---|---|
| 1 | 1KB | 0.0126305 | 7.78316 | 0.001954635 | 0.000973582 | 7.798718718 |
| 2 | 10KB | 0.0083265 | 7.75075 | 0.002922924 | 0.000968575 | 7.762967999 |
| 3 | 100KB | 0.0097285 | 7.73132 | 0.0009935 | 0.000983834 | 7.743025834 |
| 4 | 1MB | 0.009259 | 7.92991 | 0.000970244 | 0.000974536 | 7.94111378 |
| 5 | 10MB | 0.0130765 | 7.26687 | 0.001469758 | 0.001462817 | 7.282879075 |
| 6 | 100MB | 0.031743 | 7.18971 | 0.001950973 | 0.000982046 | 7.224386019 |
| 7 | 1GB | 0.008793 | 7.71865 | 0.0009675 | 0.000972033 | 7.729382533 |

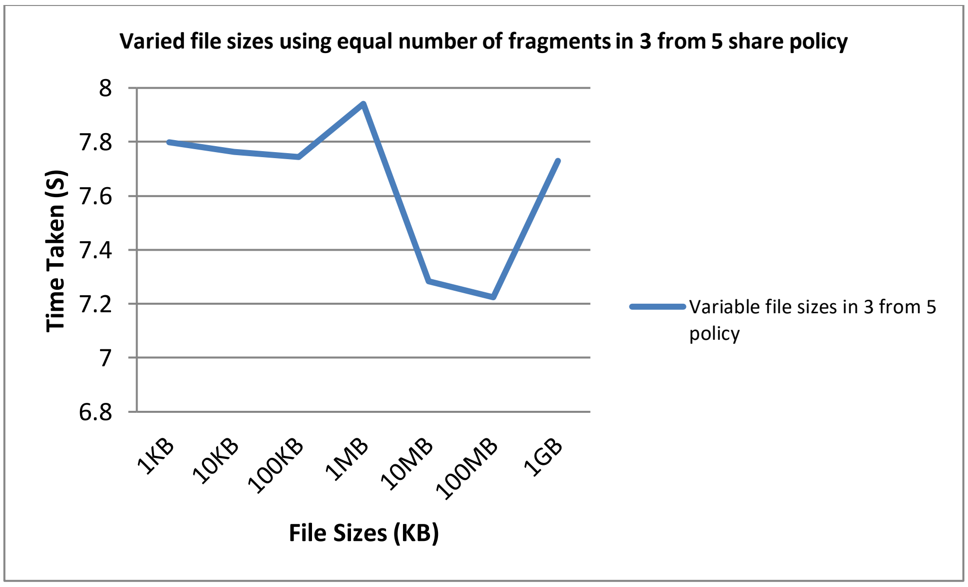


Figure 87: Varied file sizes using equal number of fragments in 3 from 5 share policy

Table 68: Varied file sizes using equal number of fragments in 4 from 5 share policy

| S/N | FileSize | KeyShaCreTime | KeyShaWriTime | ShaRecTime | SecRecTime | OverHeadCost |
|---|---|---|---|---|---|---|
| 1 | 1KB | 0.0097145 | 8.14136 | 0.002932 | 0.005845308 | 8.159851808 |
| 2 | 10KB | 0.0097625 | 7.37872 | 0.0024415 | 0.006352425 | 7.397276425 |
| 3 | 100KB | 0.0102365 | 6.89898 | 0.002934 | 0.006337404 | 6.918487904 |
| 4 | 1MB | 0.01023 | 6.77493 | 0.00097394 | 0.007277608 | 6.793411548 |
| 5 | 10MB | 0.0097235 | 8.21718 | 0.00244192 | 0.00633204 | 8.23567746 |
| 6 | 100MB | 0.0117145 | 6.63638 | 0.000980616 | 0.005851626 | 6.654926742 |
| 7 | 1GB | 0.010253 | 7.50053 | 0.000982 | 0.005854607 | 7.517619607 |

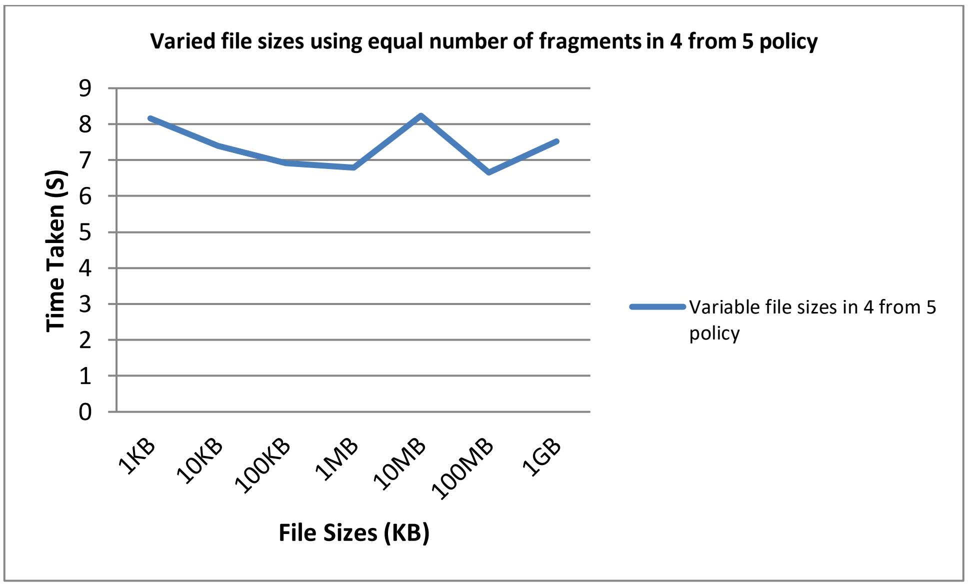


Figure 88: Varied file sizes using equal number of fragments in 4 from 5 share policy

Table 69: Varied file sizes using equal number of fragments in 4 from 10 share policy

| S/N | FileSize | KeyShaCreTime | KeyShaWriTime | ShaRecTime | SecRecTime | OverHeadCost |
|---|---|---|---|---|---|---|
| 1 | 1KB | 0.018493 | 18.631 | 0.001472185 | 0.024397969 | 18.67536315 |
| 2 | 10KB | 0.0233775 | 16.9215 | 0.002914 | 0.02342999 | 16.97122149 |
| 3 | 100KB | 0.0214355 | 17.8164 | 0.005371 | 0.02293098 | 17.86613748 |
| 4 | 1MB | 0.023038 | 18.6172 | 0.00243494 | 0.029303908 | 18.67197685 |
| 5 | 10MB | 0.0180025 | 16.3501 | 0.0024365 | 0.024405718 | 16.39494472 |
| 6 | 100MB | 0.020926 | 15.8723 | 0.0034195 | 0.023820877 | 15.92046638 |
| 7 | 1GB | 0.0248405 | 16.101 | 0.004886 | 0.024891376 | 16.15561788 |

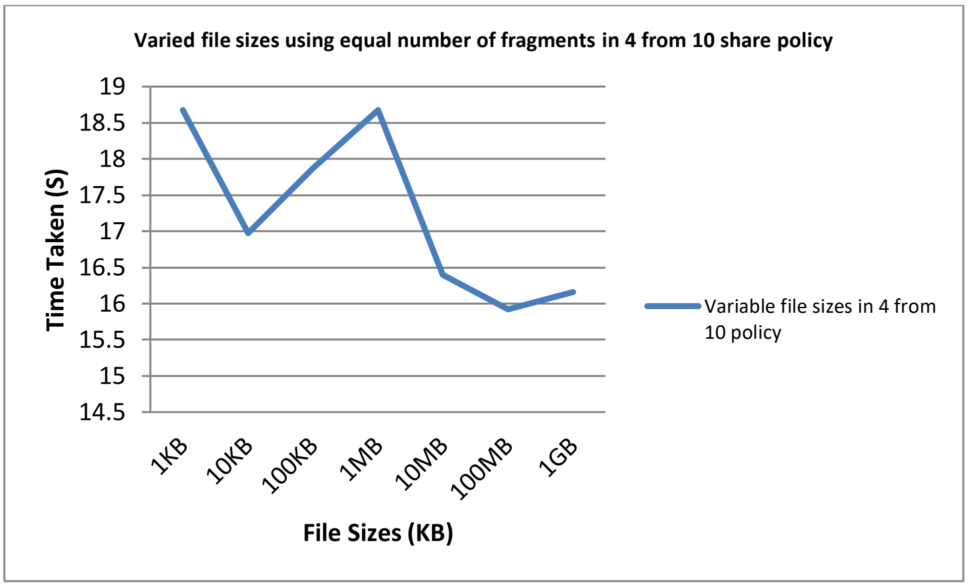


Figure 89: Varied file sizes using equal number of fragments in 4 from 10 share policy

Table 70: Varied file sizes using equal number of fragments in 6 from 10 share policy

| S/N | FileSize | KeyShaCreTime | KeyShaWriTime | ShaRecTime | SecRecTime | OverHeadCost |
|---|---|---|---|---|---|---|
| 1 | 1KB | 0.0233445 | 16.2525 | 0.0024405 | 9.392581463 | 25.67086646 |
| 2 | 10KB | 0.0238365 | 15.9982 | 0.001952566 | 9.305171847 | 25.32916091 |
| 3 | 100KB | 0.0287805 | 15.26517272 | 0.0029445 | 9.76513195 | 25.06202967 |
| 4 | 1MB | 0.0287635 | 15.6713 | 0.002930973 | 9.67675817 | 25.37975264 |
| 5 | 10MB | 0.026309 | 16.1595 | 0.001954331 | 9.628413081 | 25.81617641 |
| 6 | 100MB | 0.0287345 | 15.956 | 0.004394 | 9.275384188 | 25.26451269 |
| 7 | 1GB | 0.026781 | 15.7265 | 0.0033945 | 9.627430558 | 25.38410606 |

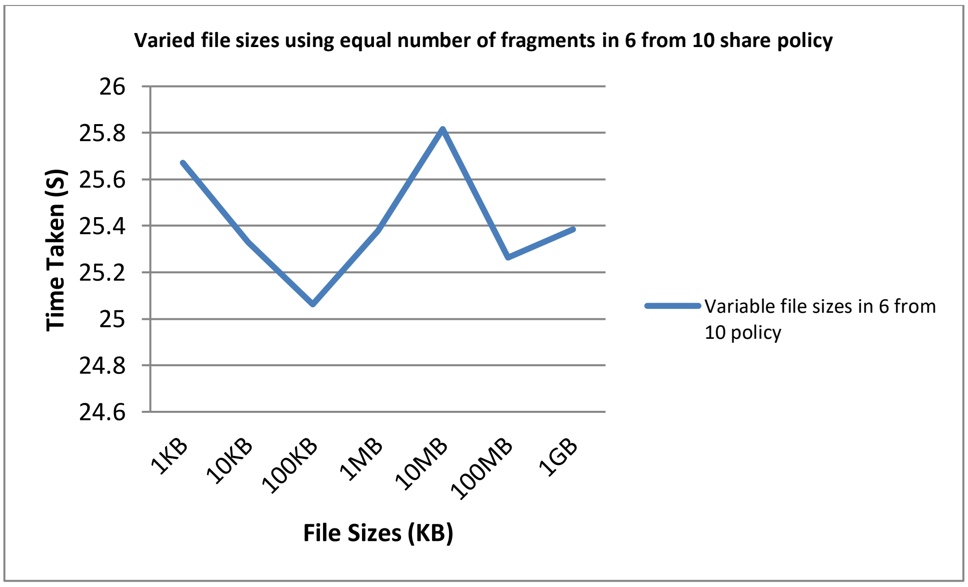


Figure 90: Varied file sizes using equal number of fragments in 6 from 10 share policy

Table 71: Varied file sizes using equal number of fragments in 8 from 10 share policy

| S/N | FileSize | KeyShaCreTime | KeyShaWriTime | ShaRecTime | SecRecTime | OverHeadCost |
|---|---|---|---|---|---|---|
| 1 | 1KB | 0.0555445 | 17.0244 | 0.0019635 | 1519.267616 | 1536.349524 |
| 2 | 10KB | 0.035603 | 15.6087 | 0.002954 | 1613.893052 | 1629.540309 |
| 3 | 100KB | 0.035248 | 16.7949 | 0.00243842 | 1467.797424 | 1484.63001 |
| 4 | 1MB | 0.031649 | 15.9978 | 0.002927824 | 1428.302855 | 1444.335232 |
| 5 | 10MB | 0.0355935 | 15.448 | 0.002912 | 1500.366265 | 1515.85277 |
| 6 | 100MB | 0.0332835 | 16.9651 | 0.004884 | 1455.750583 | 1472.75385 |
| 7 | 1GB | 0.031208 | 21.5647 | 0.013211 | 1577.407721 | 1599.01684 |

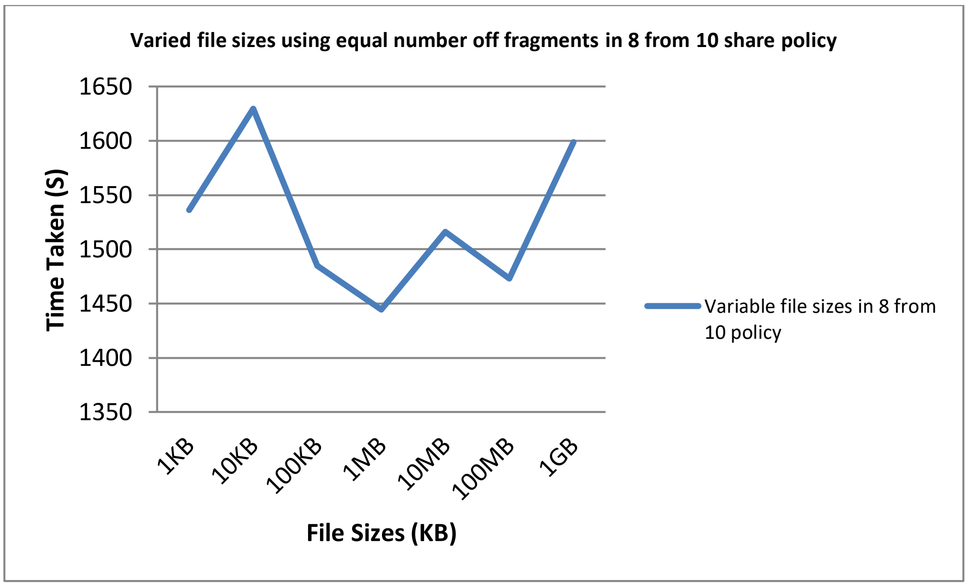


Figure 91: Varied file sizes using equal number of fragments in 8 from 10 share policy

## 12.4 Variant Three

### 12.4.1 Fragments

**File Sizes: Varied**

**Fragment Sizes: Varied**

**Secret Sharing Policy: Varied**

**Cloud Outage: Varied**

**Plot: File Sizes in KB against Time Taken in Seconds to process and combine file using varied key share policies with cloud outages.**

Table 72: Cloud outages at varied file sizes and share policies

| S/N | KeyShaFileSize | FileSplitTime | FragEncTime | FragDecTime | FileComTime | OverHeadCost |
|---|---|---|---|---|---|---|
| 1 | 1KB, 3 from 5, 1down | 0.003419161 | 0.019013 | 0.020502925 | 0.001953244 | 0.04488833 |
| 2 | 10KB, 3 from 5, 2 down | 0.002927661 | 0.0063275 | 0.043943286 | 0.001934886 | 0.055133333 |
| 3 | 1KB, 6 from 10, 3down | 0.003908277 | 0.006839 | 0.012194753 | 0.001466274 | 0.024408304 |
| 4 | 10KB, 6 from 10, 4down | 0.003423333 | 0.0068415 | 0.015063882 | 0.002443552 | 0.027772267 |

Table 73: Varied file sizes and share policies

| S/N | KeyShaFileSize | FileSplitTime | FragEncTime | FragDecTime | FileComTime | OverHeadCost |
|---|---|---|---|---|---|---|
| 1 | 1KB, 3 from 5, 1 down | 0.01074183 | 0.007808 | 0.0117203 | 0.001464128 | 0.031734258 |
| 2 | 10KB, 3 from 5, 2down | 0.003417015 | 0.005401 | 0.014142394 | 0.001951933 | 0.024912342 |
| 3 | 1KB, 6 from 10, 3down | 0.002924442 | 0.005363 | 0.014597535 | 0.001461148 | 0.024346126 |
| 4 | 10KB, 6 from 10, 4down | 0.002439857 | 0.003923 | 0.012702703 | 0.001458526 | 0.020524086 |

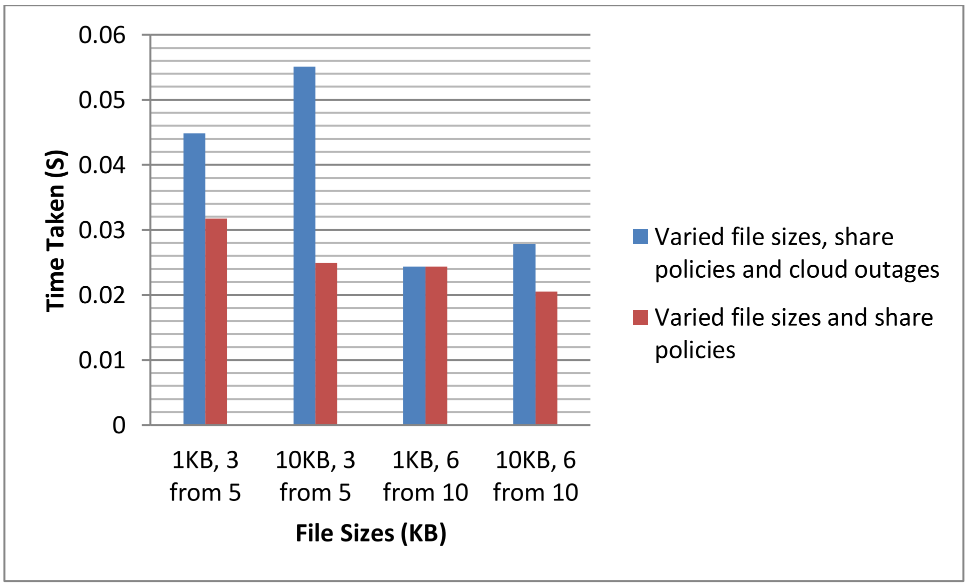


Figure 92: Comparing file processing overheads during normal situations and cloud outages

### 12.4.2 Key Share Creation and Recovery

**File Sizes: Varied**

**Fragment Sizes: Varied**

**Secret Sharing Policy: Varied**

**Plot: File Sizes in KB against Time Taken in Seconds to process and combine file using varied key share policies.**

Table 74: Cloud outages at varied file sizes and share policies

| S/N | KeyShaFileSize | KeyShaCreTime | KeyShaWriTime | ShaRecTime | SecRecTime | OverHeadCost |
|---|---|---|---|---|---|---|
| 1 | 1KB, 3 from 5, 1down | 0.0111955 | 6.5083 | 0.00098145 | 0.000971913 | 6.521448863 |
| 2 | 10KB, 3 from 5, 2down | 0.0074035 | 3.7339 | 0.000978708 | 0.000979543 | 3.743261751 |
| 3 | 1KB, 6 from 10, 3 down | 0.026288 | 13.8595 | 0.000978 | 2.049305081 | 15.93607108 |
| 4 | 10KB, 6 from 10, 4down | 0.036072 | 13.3141 | 0.001961 | 1.091316342 | 14.44344934 |

Table 75: Varied file sizes and share policies

| S/N | KeyShaFileSize | KeyShaCreTime | KeyShaWriTime | ShaRecTime | SecRecTime | OverHeadCost |
|---|---|---|---|---|---|---|
| 1 | 1KB, 3 from 5 | 0.0126305 | 7.78316 | 0.001954635 | 0.000973582 | 7.798718718 |
| 2 | 10KB, 3 from 5 | 0.0083265 | 7.75075 | 0.002922924 | 0.000968575 | 7.762967999 |
| 3 | 1KB, 6 from 10 | 0.0233445 | 16.2525 | 0.0024405 | 9.392581463 | 25.67086646 |
| 4 | 10KB, 6 from 10 | 0.0238365 | 15.9982 | 0.001952566 | 9.305171847 | 25.32916091 |

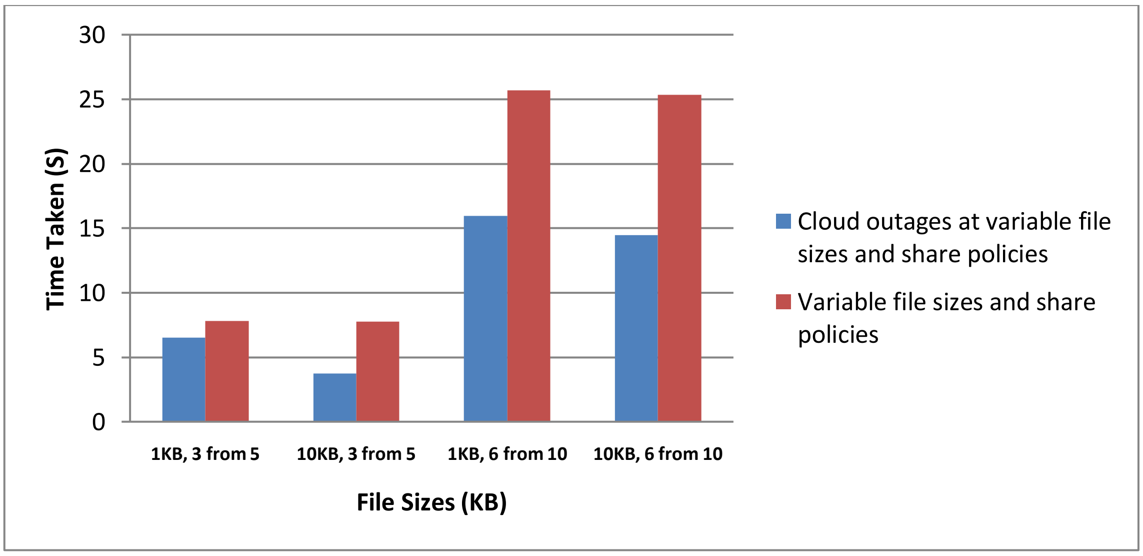


Figure 93: Comparing cloud outages at varied share policies against normal situations

# 13 Appendix D

## 13.1 Details of the experimental procedures used

Table 76: User Management Data Store

| Datetime | ID | UUID | FileName | FileSize | FileRef | Cloudlet0 | Cloudlet1 | ... | Cloudletn |
|---|---|---|---|---|---|---|---|---|---|
| Datetime$_1$ | ID$_1$ | UUID$_1$ | Filename$_1$ | Filesize$_1$ | FileRef$_1$ | FileID$_0$ | FileID$_1$ | ... | FileID$_n$ |
| .<br>.<br>. | .<br>.<br>. | .<br>.<br>. | .<br>.<br>. | .<br>.<br>. | .<br>.<br>. | .<br>.<br>. | .<br>.<br>. | .<br>.<br>. | .<br>.<br>. |
| Datetime$_n$ | ID$_n$ | UUID$_n$ | Filename$_n$ | Filesize$_n$ | FileRef$_n$ | FileID$_{n0}$ | FileID$_{n1}$ | ... | FileID$_n$ |

**Legends:**

| | |
|---|---|
| DateTime | = specifies Date and Time user details were stored. |
| ID | = contains hashed value of all user personal details |
| UUID | = Universally Unique Identifier automatically created for each user. It differentiates users and used to name file fragments when created in order of creation (UUID.1, UUID.2,..,UUID.n). |
| FileName | = specified name of file from which fragments were created |
| FileSize | = provides size of user's file |
| FileRef | = Reference number of user's file, serves as a Primary key. |
| Cloudlet0-Cloudletn | = provides FileID that reveals share storage details (cloud API, token, name of share and location in storage bucket of the cloud). |

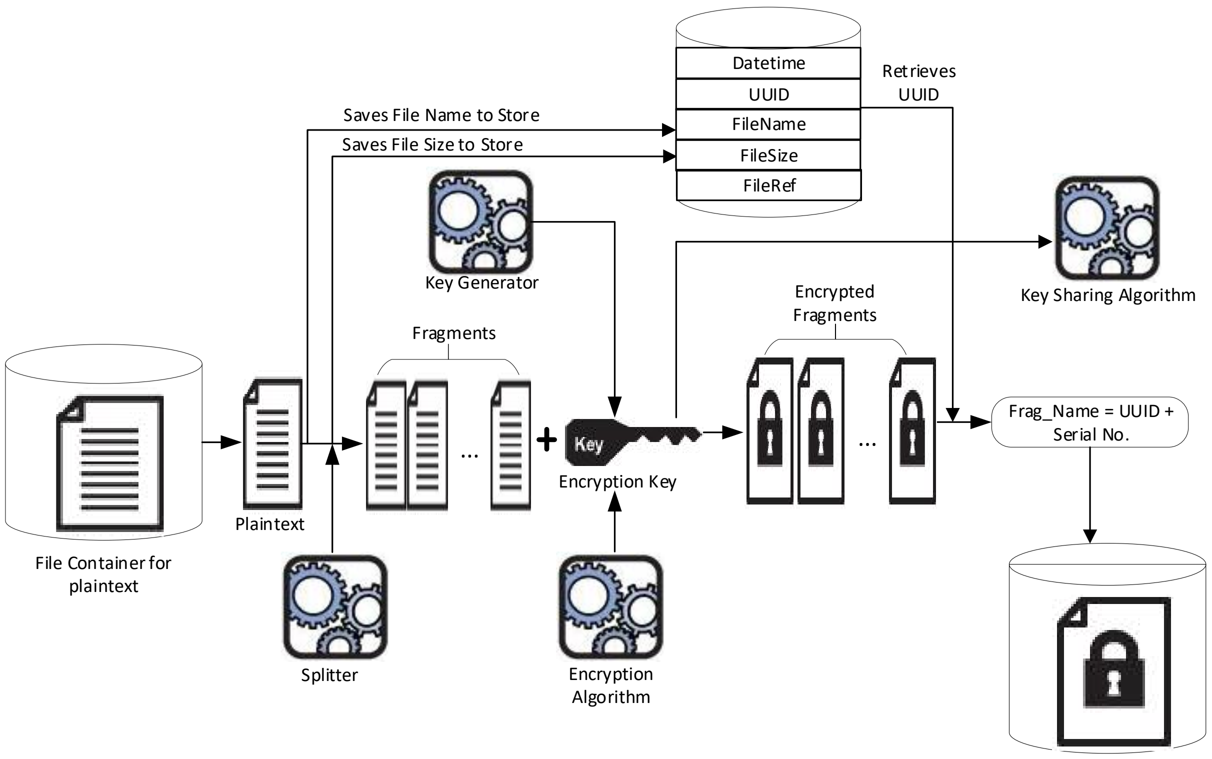


Figure 94: File Fragmentation and Encryption

## A. Processes of Share Creation

To create shares out of a secret key, we define the Threshold, M and Share size N, determine a unique Identifier, (an identifier that distinguishes each Share creation session containing values (0-9, a-z)). The essence is to establish the number of participating cloudlets and the minimum number of cloudlets that are required to reconstruct the secret key in case of share corruption or failure of retrieval from one or more cloudlets during the reconstruction phase. That is to say:

Define policy: *M-out-of-N*

Determine unique identifier: combination of alphanumeric characters

Key path = (‘path/to/keygen’)

Hash value: if the hash value chosen is 1, the secret will be hashed with SHA128, if 2, SHA256.

B. Key extraction:

Keys are automatically generated by a random number generator using,

Secret = os.urandom (BS), where BS represents block size and Secret is the encryption key.

## C. Share generation:

Using the chosen share policy's threshold, the degree of the polynomial is determined, which is the value of (*M-1*) and the number of values generated for (*X*, *A*) pair is based on *N*, and hence the values are generated using this equation [137]:

$$f(X, A) = \sum_{i=0}^{M-1} A[i].X^i \qquad \text{Equation 25}$$

*X* is the number of participants from *1* to *N*; *A* represents the values taken at random for each participant (*X*). Using this, shares are generated for each participant and the number of shares generated is based on the value of *N* as stated earlier. The secret is set at an index, where *X = 0*, as the value of *A* where *X = 0*, known as the intercept lies the secret. Hence the secret lies at *A[0] = 0*, choosing *A[1], A[2],...A[M-1]* at random corresponding shares are computed and distributed to all participants. RESCUE uses the diagram in Figure 95 to explain how each share is represented at the storage locations after computation.

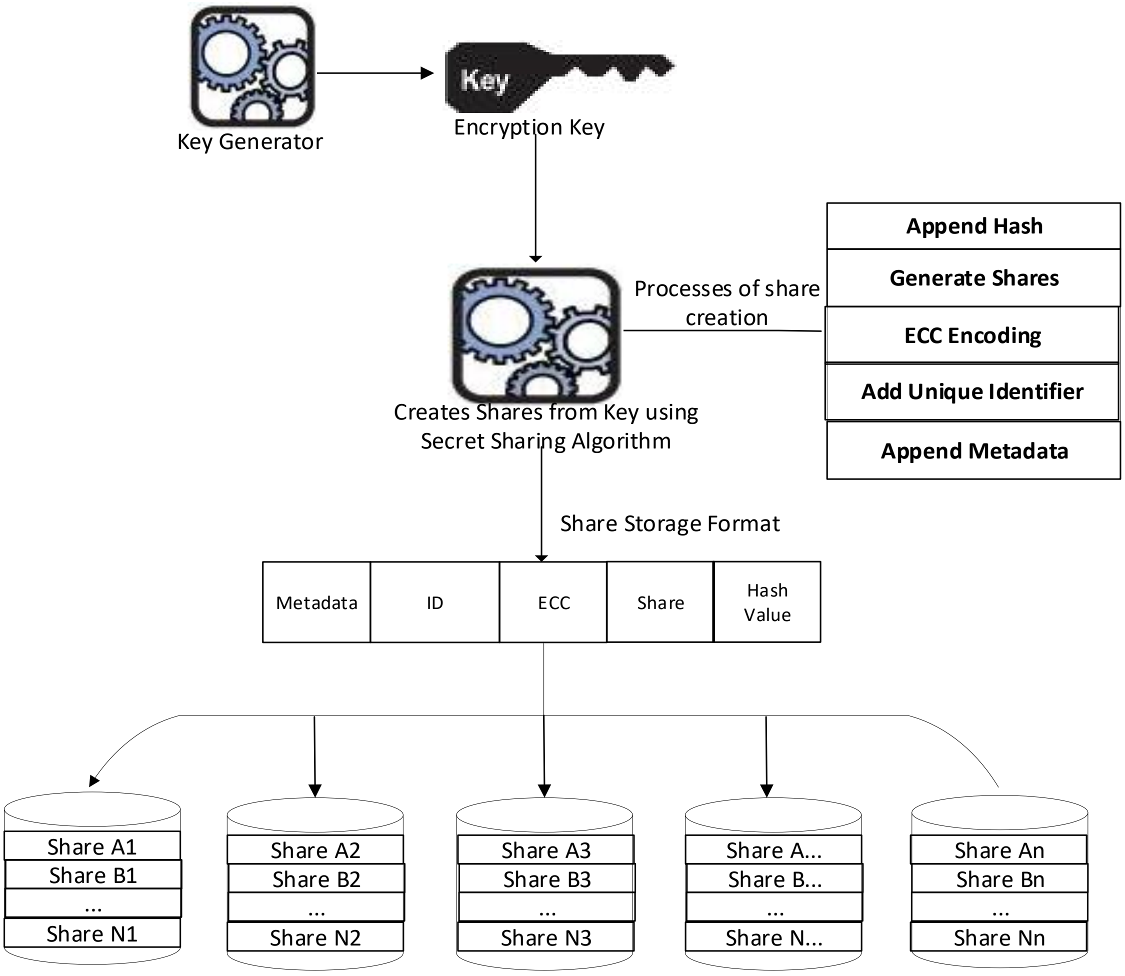


Figure 95: Key Share and Storage

## D. File Recreation

File recreation in RESCUE as shown in Figure 96 are of several stages – the user types in personal details as well as file reference number at login stage, RESCUE produces their equivalent hash values and matches them with stored values in user's metadata store. When matched values are found, the user's access is granted and a Pointer is placed on the record. First, the UUID of the user is retrieved and this is used call up all correspondent encrypted fragments in memory. Secondly, FileIDs of each participating cloudlets are retrieved and used to locate shares created out of the key for each encrypted fragment. Thirdly, each key is recovered from the participating cloudlets using Secret Share Recovering Algorithm based on a pre-determined threshold. Fourthly, each recovered key is used to decrypt correspondent encrypted fragments in sequence until all encrypted fragments are decrypted. The decrypted fragment each is written in sequence (serial numbered) until all fragments are written on top of each order and hence original file is reconstructed. Finally, the original file name with the corresponding file extension name are retrieved and used to name the reconstructed file. The file so reconstructed is checked for integrity using SHA 512 checksum, and when valid, it is moved to a designated file container. We hereby present the mathematical expression of the secret share recovery algorithm as follows:

Mathematically, *M* numbers of participants collaborate to recover the secret ***f(0)*** using Lagrange interpolation such as [137]:

$$f(0) = \sum_{i=0}^{M-1} f(x_i) \prod_{k=0,k\neq i}^{M-1} \frac{x_k}{x_k - x_i}$$ Equation 26

Ith values are the minimum number of participants that can collaborate to recover the secret, while kth values are the maximum number of participants and they are not the same. Just as in Shamir [2] authorised participants following earlier stated rules are able to recover the secret using Lagrangian interpolation once the conditions:

1. All zero elements of the array of M octets are retrieved.
2. Number of retrieved elements greater or equal to the threshold.
3. All contributed shares from participants are certified as genuine and satisfies 2 above.

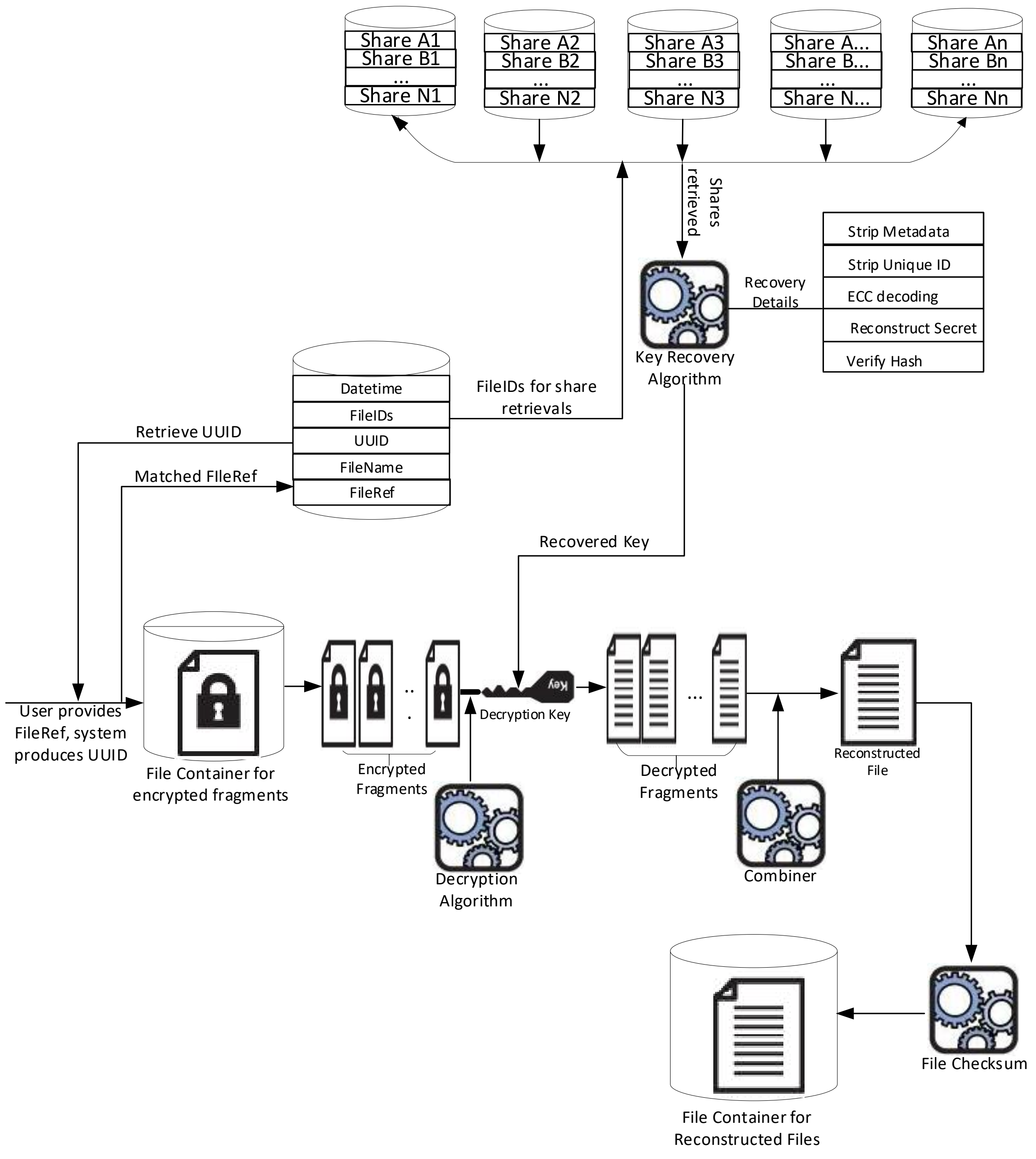


Figure 96: Key recoveries and file reconstruction

## E.Cloud Behavioural Computation

After every operation, the user is presented with two different computational options – Capacity Measures of each participating cloudlet or Overall Cloudlets Performance during each operation. The essence is to give the user options of perusing behaviours of each cloudlet during the operation such as its Latency (ms), Speed (bps), Throughput (bps) and Availability. While in the other hand, the overall performance of all cloudlets in the areas of Elapsed Time (sec), Packet Loss, Average Round Trip Time (ms), Speed (bps), Download Bandwidth (bps), Upload Bandwidth (bps), Latency (ms), and Throughput (bps) for comparative analysis. We will therefore treat this section by looking at these measures in details as designed.

## III. Capacity Measures of clouds

Below is a table showing the Capacities, Evaluation Metrics, Formulae, Units and Tools for measurement.

Table 77: Evaluation Metrics

| Capacity | Metrics | Formulae | Unit | Tool |
|---|---|---|---|---|
| Availability | Packet Loss Frequency | Packet Loss Per Unit Time | Bits per second (bps) | Pingparser |
| Latency | IP Transfer Delays | Propagation Delay + Serialisation delay | Milliseconds (ms) | SpeedNet |
| Transaction Speed | Max. No. of Transfer Session | Length of File Over Time | Meters per Second (mps) | SpeedNet |
| Throughput | Volume of processed data | File Size Over Time | Ops/Sec (bps) | SpeedNet |

## IV. Overall Performance Measures

Overall performance extends some measures by including more metrics that shows comparatively cloudlets performance at a glance by considering their different capacities such as: Elapsed Time (s), Packet Loss, Average Round Trip Time (ms), Speed (s), Download and Upload Bandwidths (bps), Latency (bps) and Throughput (bps).

Table 78: Overall Performance Evaluation Metrics

| Capacity | Metrics | Formulae | Unit | Tool |
|---|---|---|---|---|
| Elapsed Time | Time Taken for action | End Time - Start Time | seconds | Clock Time |
| Packet Loss | Packet differentials | Received – Transmitted packets | bps | SpeedNet |
| Average Round Trip Time | Mean Round Trip Time | Total RTT/Number of Round Trips | ms | Pingparser |
| Speed | Max. No. of Transfer Session | Length/Time taken | mps | Pingparser |
| Download Bandwidth | Total download traffic carrying capacity | Volume of data transmitted between two points per second | bps | SpeedNet |
| Upload Bandwidth | Total upload traffic carrying capacity | Volume of data transmitted between two points per second | bps | SpeedNet |
| Latency | IP Transfer Delays | Propagation Delay + Serialisation delay | ms | SpeedNet |
| Throughput | Volume of Processed Data | File Size Over Time | bps | SpeedNet |

At every operation of RESCUE, measures are taken to ascertain several behaviours of participating cloudlets so as to help set up benchmarks of capacities of participating cloudlets for every file size against share policy in relation to accrued performance overheads such as latency, rate of packet loss, transaction speed and throughput. With this one can say using a certain file size in conjunction with a fragment size and share policy, the system is capable of having certain overheads accrued and hence prediction of future behaviours, as well as potential dangers, can be made using available results. With regards to this thesis' proposed future works, self-organisation can be activated to avert any impending danger so as to keep the system consistently available. These behaviours are calculated through capacity measures as provided in the above tables.

## F. Agent

The activities to be performed by the proposed Agent using Linear Regression model in Machine Learning as well as our Mathematical model to help predict the expected outcome of each cloudlet's behaviour given a certain condition and inputs from the user are elaborated in section 8.4 of Chapter 8. The works of the Agent is to receive data from the calculated cloudlets capacities from several Microsoft Excel files and use same based on the metrics to 1. Establish a relationship between capacity's variables and 2. Use the same to predict future results. When such results so predicted portends potential danger, the Agent retrieves shares posted to such cloudlet(s) and send same to better behaved cloudlets without giving any cloudlet shares exceeding or equals to the minimum thresholds used in such operation.