

SECURING CLOUD DATA ON MULTIPLE INFRASTRUCTURE USING ERASURE CODING, DISPERSAL TECHNIQUE AND ENCRYPTION

KNUST

By

Frimpong Twum (CCNA, HND Engnr., BSc. (Hons) Engnr., MSc. Engnr., MSc. IS,)

A Thesis submitted to the Department of Computer Science, Faculty of Physical Sciences, College of
Science, KNUST, in partial fulfilment of the requirements for the degree of

DOCTOR OF PHILOSOPHY IN COMPUTER SCIENCE

December, 2017

DECLARATION

“I hereby declare that this submission is my own work and that, to the best of my knowledge and belief, it contains no material previously published or written by another person nor material which to a substantial extent has been accepted for the award of any other degree or diploma at Kwame Nkrumah University of Science and Technology, Kumasi or any other educational institution except where due acknowledgement is made in the thesis.”

Frimpong Twum (PG1547413)

Student Name & ID

Signature

Date

Certified by:

Dr. J. B. Hayfron-Acquah

Supervisor

Signature

Date

Certified by:

Prof. William W. Oblitey

Supervisor

Signature

Date

Certified by:

Dr. Michael Asante

Head of Department

Signature

Date

ABSTRACT

Cloud computing is a technology that has come to save organisations from investing in and owning high cost IT infrastructure including its management and maintenance. The technology enables an organisation to outsource its IT needs to the care of a remote third party Cloud Service Provider (CSP) while focusing on its core business processes. It enables the usage of IT resources remotely as a service on subscription basis at a per usage fee on demand. The service models available are Infrastructure as

a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). These service models are deployed in one of four cloud deployment models as Public, Private, Community or Hybrid cloud. Despite the technology's numerous benefits, it also poses serious security threats to vital business data assets as the subscriber has to surrender control over its management and maintenance to a remote CSP. The threats include: the CSP using the data for their own gains, the location of the data not known to the subscriber, the ownership of the data (for example, on contract termination or in the event of conflict or dispute?), and also the subscriber not knowing who has unauthorised access to their data resource. The challenge therefore, is how to create a secure and vigorous data security solution that can mitigate these threats and alleviate the cloud subscriber fear to freely enjoy using cloud computing services. Hence, this study proposes and implements a Six-level Cloud Data Distribution Intermediary (CDDI) Framework that enables the cloud subscriber to effectively secure its data against these threats. The framework employs Erasure Coding (based on the Galois Field Theory and Reed Solomon Coding), Data Dispersion technique with a proposed Transposition Encryption technique based on Rubiks cube transformation. In addition, it also uses this study's proposed Erasure Coding technique based on checksum dubbed "*Checksum Data Recovery*". The CDDI framework implemented on the cloud subscriber's gateway system encrypts and splits the subscriber's data into chunks of data fragments and distributes them randomly to the subscribers selected multiple CSP storage infrastructures. This alleviates threats of data usage, location, ownership, and access, identified. By employing design research methodology, the CDDI framework is developed into software following a Plan-Driven Incremental software development approach. The system dubbed 'SecureMyFiles (SMF)' was developed in an experimental lab set-up using JAVA, SQL, and PHP. The SMF system provides users a choice of selecting one of four data priority levels (Low, Normal, Important, Critical) at the time of uploading data resources to the cloud. The priority level selected determines the uploading and downloading process the system uses, the amount of data that can be recovered in the event of data corruption and the performance during recovery. The security strength of the SMF system in relation to assuring the cloud subscriber of the Confidentiality, Integrity, and Availability of their data was found to be much stronger than the existing direct architecture model provided by DropBox, Box, Google, Backblaze B2, or the indirect architecture model provided by CASB/SECaaS providers. This is because with the SMF system the subscriber data does not reside with one single provider but distributed across many providers distributed storage infrastructure.

TABLE OF CONTENTS

DECLARATION.....	i
ABSTRACT	ii
TABLE OF CONTENTS	iii
LIST OF TABLES.....	vi
LIST OF FIGURES	vii
LIST OF LISTINGS	x

LIST OF EQUATIONS.....	xiii
DEDICATION.....	xiv
ACKNOWLEDGEMENTS.....	xv
CHAPTER 1	1
1. INTRODUCTION.....	1
1.0. Background of Study	1
1.1. Overview of Cloud Computing	2
1.2. Problem Statement.....	3
1.3. Aim of Research	6
1.4. Specific Research Objectives:	6
1.5. Research Questions.....	6
1.6. Justification of choice of the study	6
1.7. Methodology.....	7
1.8. The study organisation.....	9
CHAPTER 2	9
2. LITERATURE REVIEW	9
2.0. The Concept of ‘Cloud’ Computing.....	9
2.1. Existing measures for securing cloud resources	19
2.2. Cloud computing challenges	22
2.3. Summary of Cloud computing security issues	23
2.4. Some examples of Network Attacks on Cloud Computing.....	23
2.5. An evaluation of standard counter-security measures used by CSP’s.....	26
2.6. Ensuring Data Security	27
2.7. Evaluation of Encryption Algorithms.....	29
2.8. Reed Solomon Coding.....	39
2.9. Review of Existing Related Cloud File Storage Systems.....	56
2.10. Review of Existing Cloud Storage Security Architecture	63

CHAPTER 3	67
3. METHODOLOGY	67
3.0. Introduction	67
3.1. The Proposed Architecture	68
3.2. Cloud Data Distribution Intermediary (CDDI)	69
3.3. File Splitting And Erasure Protection Sub-Module.....	77
3.4. Metadata Module	92
CHAPTER 4	93
4. IMPLEMENTATION	93
4.0. Introduction	93
4.1. SecureMyFiles System	93
4.2. The Login Module	93
4.3. File Upload Module.....	94
4.4. Implementation Of The Hashing Method.....	95
4.5. Implementation Of The Proposed Transposition Cipher Algorithm Based On Rubik's Cube Transformation.....	95
4.6. Implementation Of The Data Distribution Module	99
4.7. Reed Solomon Coding.....	99
4.8. Implementation Of File Splitting And Erasure Protection Module (FSEPM) Using Java	107
4.9. Implementation Of The Checksum Data Recovery Technique.....	116
4.10. Implementation Of The Data Dispersal Technique (The Shuffling Method).....	125
4.11. Implementation Of SMF System's Metadata	126
4.12. Shards Upload Module	127
4.13. File Download Module.....	128
4.14. Reed Solomon Decoding Process.....	130
4.15. Checksum Data Recovery Decoding	133
4.16. Web Server	146

CHAPTER 5	149
5. TESTING, RESULTS AND DISCUSSIONS	149
5.0. Testing and Results.....	149
5.1. Cloud Providers	149
5.2. File Uploading Sequence.....	150
5.3. File Downloading Sequence.....	161
5.4. The Proposed Cloud Data Distribution Intermediary (CDDI) Framework.....	170
5.5. Discussion of how the proposed system compares with existing related systems	170
CHAPTER 6.....	173
6. FINDINGS, CONCLUSIONS AND RECOMMENDATIONS	173
6.0. Findings	173
6.1. Conclusions	180
6.2. Recommendations	182
REFERENCES	182
APPENDIX 1	191
APPENDIX 2	202
APPENDIX 3	213

LIST OF TABLES

Table 2.1 - Summary of the cloud deployment models	18
Table 2.2 - Representation of finite field $GF(2^3)$ as powers of 2 using the prime number 11	44
Table 2.3- Representation of finite field $GF(2^3)$ as powers of 2 using the prime number 13	45
Table 2.4 - Addition in $GF(8)$ mod 2	48
Table 2.5 - Multiplication in $GF(8)$ mod 2	49
Table 2.6 - Lookup Table Using RS Codeword of length 7	53

Table 5.1 - Encoding of 256-byte file with 32 data shards and 8 parity shards.....	147
Table 6.1 - How the proposed system address issues of Confidentiality, Integrity, Ownership, Availability and Authentication	Error! Bookmark not defined.
Table 6.2 - How the CDDI framework addresses other cloud security issues.....	Error! Bookmark not defined.
Table 6.3 - Comparison of the CDDI framework with existing architectures.....	Error! Bookmark not defined.

LIST OF FIGURES

Figure 2.1 – NIST visual model of cloud computing	12
Figure 2.2a – Private Cloud of a company with 3 business Units	12
Figure 2.2b – Public Cloud provider with 3 business units	12
Figure 2.3- Architecture of cloud service models	13
Figure 2.4 - Cloud services types and resource control	14
Figure 2.5 - Concept of encryption and decryption	30
Figure 2.6 - Transpositional Cipher	33
Figure 2.7 - Route Cipher	33
Figure 2.8 - Rail Fence Cipher Encryption	34
Figure 2.9 - Rail Fence cipher Decryption.....	35
Figure 2.10 - Components of Conventional Encryption Algorithms	36
Figure 2.11 – GFS Architecture	57
Figure 2.12 – GFS Architecture	58
Figure 2.13 – Apache Hadoop Architecture	60
Figure 2.14 – Backblaze B2 System	62
Figure 2.15 – Backblaze B2 System Architecture	63
Figure 2.16 - Direct Model of Subscriber-CSP interaction	64
Figure 2.17 - Security Architecture of Dropbox	65
Figure 2.18 - Indirect Model of Subscriber-CSP Interaction Using Cloud Access Security Broker ...	67

Figure 2.19 - Example Implementation of Cloud Access Security Broker via Cloud Proxy	67
Figure 3.1 - Proposed Model for Cloud Data Storage Using CDDI	69
Figure 3.2a - Six faces of the Rubik's Cube	71
Figure 3.2b - Six faces of the Rubik's Cube initialized with data	71
Figure 3.3a - Cube rotation pattern	74
Figure 3.3b - Cube rotation pattern with data	74
Figure 3.4 - Flow diagram for the algorithm	79
Figure 3.5 - Overall architecture of the proposed Checksum Data Recovery program	83
Figure 3.6 - Modular representation of data	83
Figure 3.7 - Module diagram of the proposed Checksum Data Recovery Program	84
Figure 3.8 - Architecture of the Data Module	85
Figure 3.9 - Architecture of the ComputeParities Module	86
Figure 3.10 - Architecture of the LocateError Module	87
Figure 3.11 - Flow diagram for the Checksum Data Recovery program.....	88
Figure 3.12 - Activity diagram of the Checksum Data Recovery program	89
Figure 3.13 - Sequence diagram of the Checksum Data Recovery program	89
Figure 3.14 - Class diagram for the Checksum Data Recovery program	90
Figure 4.1 - Components of SMF System	92
Figure 4.2 - File Upload Module	94
Figure 4.3 - Splitting of file and computation of parity.	114
Figure 4.4 - The formation of shards from the modules of the CDR using a 512 byte file.	127
Figure 4.5 - File Download Module	127
Figure 4.6 - Database schema	1400
Figure 5.1 - User Interface for selecting and configuring file for upload	143
Figure 5.2 - User Interface for choosing a file for upload	144
Figure 5.3 - Result from hashing file name.	145
Figure 5.4 - Result from encrypting file content	146
Figure 5.5 - Encoding Polynomial Coefficients for Parity sizes from 1 up to 32	146
Figure 5.6 - Encoding Polynomial Coefficients for Parity size of 32	146
Figure 5.7 - A view into a folder showing the shards of a file after splitting	149
Figure 5.8 - Result from file name shuffling	150
Figure 5.9 – A successful file upload choosing the Normal file priority level	150
Figure 5.10 – A successful file upload choosing the Critical file priority level	150
Figure 5.11 - Content of Dropbox account showing some of the shards from a file whose name has	

been obfuscated.	153
Figure 5.12 - Content of Box account showing some of the shards from a file whose name has been obfuscated.	
153 Figure 5.13 - Interface of SMF application showing a list of uploaded files	154
154 Figure 5.14 – successful file reconstruction during download for 24 corrupted shards with Normal option.....	154
Figure 5.15 – successful file reconstruction during download for 48 corrupted shards with Normal option.....	154
Figure 5.16 – failed file reconstruction during download for more than 48 corrupted shards with Normal option.....	154
Figure 5.17 – successful file reconstruction during download for 72 corrupted shards with Important option.....	154
Figure 5.18 – failed file reconstruction during download for more than 72 corrupted shards with Important option.....	154
Figure 5.19 – successful file reconstruction during download for 4 corrupted shards with Critical option.....	154
Figure 5.20 – successful file reconstruction during download for 8 corrupted shards with Critical option.....	154
Figure 5.21 – successful file reconstruction during download for 12 corrupted shards with Critical option.....	154
Figure 5.22 – failed file reconstruction during download for more than 12 corrupted shards with Critical option.....	154
Figure 5.23 – Proposed Six-level Cloud Data Distribution Intermediary (CDDI) Framework.....	163

LIST OF LISTINGS

Listing 3.1 – Setting up a minimum padding cube	79
Listing 3.2 – Clockwise rotations in the three planes of the cube	79
Listing 3.3 - Algorithm to generate the coefficients of the encoding polynomial	79
Listing 3.4 - Pseudocode for polynomial addition	80
Listing 3.5 - Pseudocode for polynomial multiplication	80
Listing 3.6 - Pseudocode for polynomial division	81
Listing 3.7 - Pseudocode for shifting a polynomial by a number of degrees	81
Listing 4.1 - getHash method to generate the hash of a string	94
Listing 4.2 – Function to initialise the Rubik’s Cube	95
Listing 4.3 – Implementation of the rotation function in pseudocode	96
Listing 4.4 - Cube constructor code in Java	97
Listing 4.5 – Generation of Rotation Sequence	97
Listing 4.6 - Method to convert a key to a rotation sequence	97
Listing 4.7 - Method to encrypt a file using the Rubik’s cube	98
Listing 4.8 - Method to decrypt a file using the Rubik’s cube	98
Listing 4.9 - Pseudocode for generating the Galois Field elements.	101
Listing 4.10 - Pseudocode for performing addition and subtraction in the Galois Field	101
Listing 4.11 - Pseudocode for performing multiplication in the Galois Field	102
Listing 4.12 - Pseudocode for performing division in the Galois Field.....	102
Listing 4.13 - Pseudocode for polynomial multiplication	105
Listing 4.14 - Pseudocode to generate the Reed Solomon Encoding Polynomial	106
Listing 4.15 - Pseudocode for encoding a polynomial using an RS Encoding Polynomial	106
Listing 4.16 - Snippet of code to show how the data and parity shard counts are set based on the priority setting	108
Listing 4.17 – Generation of the Galois Field Elements in Java	109
Listing 4.18 - Implementation of addition in the Galois Field	109
Listing 4.19 - Implementation of multiplication in the Galois Field	109
Listing 4.20 - Implementation of division in the Galois Field	110
Listing 4.21 - Method for accessing the log of a field element	110
Listing 4.22 - Method for accessing the exponent of two in the Galois Field	110
Listing 4.23 - Polynomial class constructor	110
Listing 4.24 - Implementation of Polynomial Addition.....	111

Listing 4.25 - Implementation of Polynomial Multiplication	111
Listing 4.26 - Implementation of Polynomial Division	112
Listing 4.27 - Implementation of the Polynomial Modulus	112
Listing 4.28 - Code for generating the Encoding Polynomial	113
Listing 4.29 - Java implementation of file encoding process using Reed-Solomon Coding	115
Listing 4.30 – Function to read from a file into a three-dimensional array	116
Listing 4.31 – reading file to 3-dimensional array	117
Listing 4.32 – Function to compute the module’s parity	117
Listing 4.33 – Function to compute row and column parities	118
Listing 4.34 – Computation of module parity	119
Listing 4.35 – Computation of row parity and column parity	119
Listing 4.36 – Population of the parity object.....	120
Listing 4.37 – Writing parity data to file	121
Listing 4.38 – Function to split file into shards	121
Listing 4.39 – Splitting data into 16 shards	121
Listing 4.40 - Code for implementation of the data dispersion technique (the shuffling method)	124
Listing 4.41 - Instantiation of a metadata file object	125
Listing 4.42 - Method to write metadata string to file	125
Listing 4.43 - Snippet of code showing file upload to Dropbox and Bo	126
Listing 4.44 -Method to read text from a file to a string.....	128
Listing 4.45 - Snippet of code showing how the system creates a metadata object from a file and accesses the list of destinations from the object	128
Listing 4.46 - Function to determine data corruption in a codeword.....	128
Listing 4.47 - Function to construct the syndrome polynomial	129
Listing 4.48 - Function for determining error locator and error magnitude polynomials	129
Listing 4.49 - Function to determine specific error locations	130
Listing 4.50 - Function to implement the Forney Algorithm to determine the magnitude of an error	130
Listing 4.51 - Function to correct errors in a Reed Solomon codeword	131
Listing 4.52 – Reading file shards to form a data array	131
Listing 4.53 – Reading parity data from files	132
Listing 4.54 – Checking modules for errors	133
Listing 4.55 – Error location within the module	136

Listing 4.56 – Error correction	139
Listing 4.57 – writing data from the three-dimensional array to file	139

KNUST



LIST OF EQUATIONS

Equation 3.1	77
Equation 3.2	77
Equation 3.3	77



DEDICATION

This study is first dedicated to God for his mercies and guidance. Secondly, this study is dedicated to my family and colleagues at work. Finally, the study is dedicated to all members of the ‘Club B’ fraternity.

KNUST



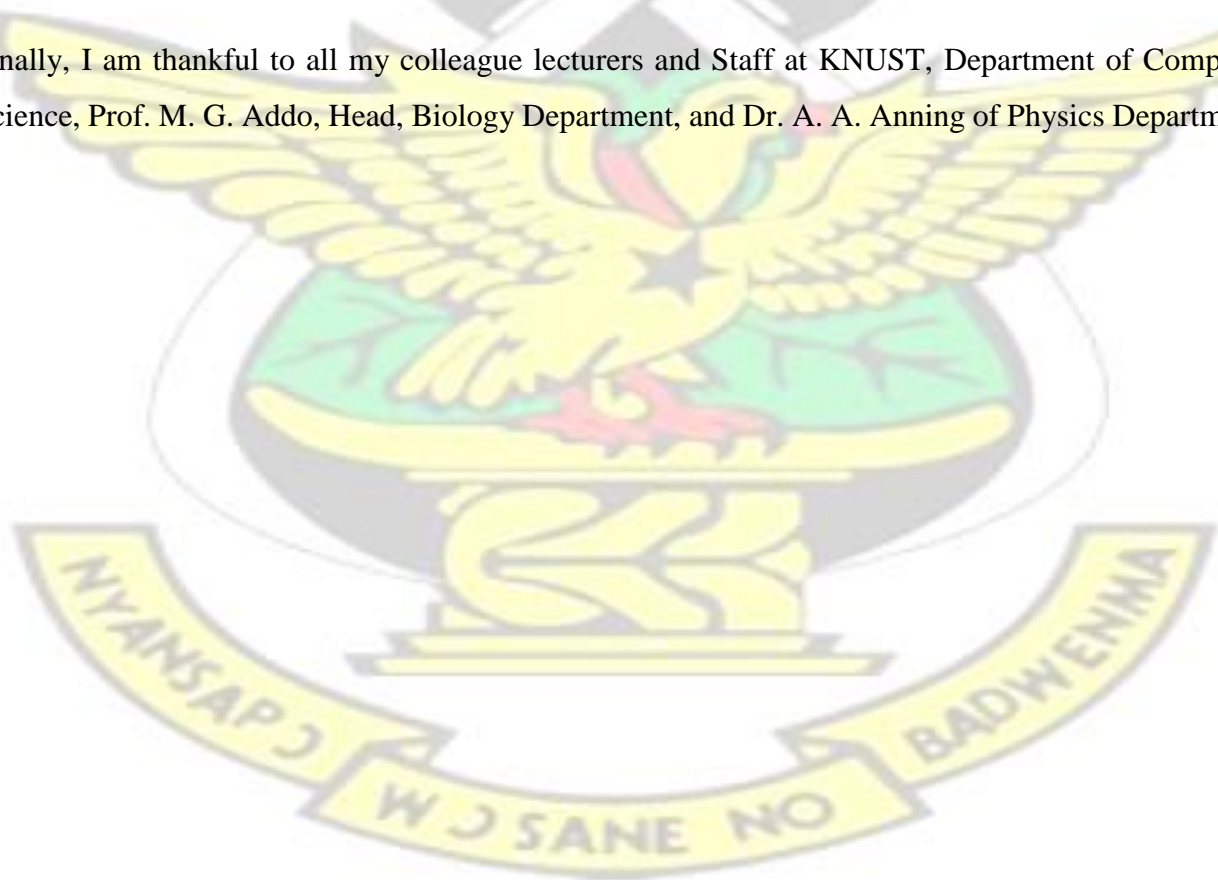
ACKNOWLEDGEMENTS

My foremost appreciation goes to God the omnipotent for giving me strength to complete this study. Secondly, I am most thankful to my supervisors, Dr. James B. Hayfron-Acquah and Prof. William W. Oblitey for their immense contribution and guidance to conduct this study and also steering it to the end. I am most grateful to them.

Special appreciation goes to Dr. R. K. Boadi and Mr. Wofa Adu Gyamfi of Mathematics department, Mr. Emmanuel Ofori Oppong, Mr. Nathaniel Frimpong, Mr. William Morgan-Darko, Mr. Yaw Darkwa, Mr. Samuel Acheampong, Mr. Dawuda Ahmed, Mr. Daniel Anane Agbemava, Mr. Benjamin Odoi Lartey, Mr. Benjamin Tei Partey, and Mr. Nuku Atta Kordzo Abiew, all of the Computer Science department for their priceless contributions. This work could not have been possible without their support.

Particular thanks also go to Prof. Kwesi Obiri Danso, Vice Chancellor of KNUST, Prof. (Mrs.) Ibok Oduro, Provost, College of Science, KNUST, and Prof. S. K. Amponsah, Dean, Faculty of Physical and Computational Sciences, KNUST, for their encouragements, advice and support.

Finally, I am thankful to all my colleague lecturers and Staff at KNUST, Department of Computer Science, Prof. M. G. Addo, Head, Biology Department, and Dr. A. A. Anning of Physics Department.



CHAPTER 1

INTRODUCTION

1.0. Background of Study

The need for Information Technology (IT) in industry has never been as high as today. However, organisations budget for IT is much stringent than ever. Businesses today invest in Information Technology and Information Systems (IS) to achieve one or several of the following strategic objectives: Improve decision making process; Survival; Customer and Supply Intimacy; Competitive Advantage over rivalry; New product, services and business model; and Operational excellence (Laudon and Laudon, 2010). To be competitive in this era, organisations ought to be strategic in their investments in IT and IS to ensure it supports key business processes that deliver the required outcome. This challenge has necessitated that organisations always find and adapt the most efficient and cost effective IT and IS solution. As a result, businesses today and even individuals are increasingly moving away from owning and managing their own IT infrastructures to using Cloud Computing technology which provides them with the ability to outsource their IT needs for example, Infrastructure as a service (IaaS), Platform as a Service (PaaS), or Software as a Service (SaaS) to the care of a remote third party provider on a subscription bases at a per usage fee on-demand (Khan and Yasiri, 2016).

Using Amazon Cloud Computing Infrastructure for instance, SaaS providers avoid having to invest heavily in IT infrastructure costs (for example, owning servers and setting up of data centres). SaaS is a software deployment model that enables the distribution of software over the Internet for subscribers' utilisation (Amazon Web Services, 2010). SaaS, which provides remote software solution to tenants (individuals or organisations) on either shared or un-shared environment on demand is what is usually referred to as cloud computing (Barry, 2000). SaaS applications offer users a centralised networked data storage facility that is hosted, maintained and managed by the Cloud Service Provider (CSP). Although cloud computing comes with many benefits, adopting cloud computing technology also means relinquishing control over key business data assets to a third party, the CSP. This is a cause of anxiety to the cloud customer and hence raises an issue of trust which is a major concern frustrating SaaS adoption.

The challenge therefore is how to create a secure and vigorous data architecture that can deal with both internal and external threats posed to the cloud tenant data resources and also satisfy subscribers legitimate concern of losing control over key business data asset to a third party CSP (Chong et. al., 2006; Rao and Selvamani 2015) - the focus of this research. SaaS incorporates IaaS and PaaS.

1.1. Overview of Cloud Computing

The National Institute of Standard 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 other computing services) that can be rapidly provisioned and released with minimal management effort or service providers interaction” (Mell and Grance, 2011).

In other words, Cloud Computing is simply the delivery of computing services over the Internet.

Cloud Computing allows organisations or individuals to use software, hardware, and data storage facilities that are hosted and managed by third parties at remote locations. It is a technology with the potential to enhance collaboration, agility, scalability, availability of computing resources, and also give organisations or individuals the added benefit of reducing their Information and Communications Technology (ICT) cost through optimised and efficient computing (Khan and Yasiri, 2016). Examples of existing cloud services include:

- Online file storage services such as Dropbox, Google Drive
- Social networking sites for example Facebook
- Online business applications such as online salary processing systems

Cloud computing give user's accessibility to their data resources from any location where they have access to the Internet and this enhances productivity and work efficiency. Cloud computing gives individuals or organisations the ability to share resources as data storage space, networks, computing processing power, and specialised corporate hardware and software applications. Music, video clips, photographs that people use to keep on their own personal computers and other mobile computing storage internal devices are now all been stored on servers owned and managed by third party CSP's such as Amazon, Google, Facebook.

With cloud computing, a subscriber has to sign-up for a service model (SaaS, PaaS, or IaaS) and subscribe to one of the cloud deployment models (private cloud, public cloud, community cloud, or hybrid cloud) on a subscription-based payment agreement with a CSP to access the storage space or other computing resources required on-demand (Goswami and Singh, 2012).

'Public' cloud computing services (public SaaS) has been in use for years for example via client webmail services such as Yahoo, Hotmail, Gmail, and also most recently via online data storage

services as DropBox, Google drive, OneDrive, and Box. These services have generally been offered at no cost to the subscriber although they are paid for in disguise for instance via advertisements presented to the subscriber online while using the services, or via the selling of subscriber's personal data and online surfing behaviour patterns to interested organisations (Barry, 2000).

Client webmail software is used to access electronic mails stored in a client's email storage accounts located on remote systems that usually belongs to other people or organisations (known or unknown) and are also stored at remote unknown locations. The remotely known or unknown individual or organisation (SaaS cloud provider) is entrusted with taking care of the subscriber's software and hardware needs, and also most importantly taking care of the subscriber's vital data assets which may include key business financial data, trade secrets, employee records, supplier's contacts, product lines, and customer's data.

This poses serious security threats to the cloud subscriber in terms of data security as the subscriber relinquishes control over the management and maintenance of the data to a third party CSP who might use the data for other purposes for which the data owner permission has not been sought.

Hence, there is a challenge with the technology ability to ensure data confidentiality, data integrity, and even data availability (Rao and Selvamani, 2015; Ali et. al., 2015). Therefore, the question of the security of the data and other resources outsourced for cloud storage remains to be a major hindrance to the technology widespread adoption (OpenCirrus, 2017).

1.2. Problem Statement

Cloud computing technology is an invention in the ever changing computing technology that has come to save organisations from setting up, owning, and maintaining high cost computing equipment and other ICT infrastructure. Benefits includes cost savings (in terms of hardware, software, personnel, etc.), ability to access resources from anywhere at any-time provided there is an Internet enabled device and connectivity to the Internet, and paying per usage among others. Cloud computing gives users huge storage capacity via storage facilities hosted on the Internet that are usually owned and managed by third party Cloud Service Providers (CSP's). These storage facilities usually are publicly accessible referred to as Public Cloud, or may be configured for an individual subscriber's private use referred to as Private Cloud, or configured explicitly for a group of organisations usage referred to as Community Cloud, or may be a composite of two or more of the specific cloud deployment models referred to as Hybrid Cloud. The CSP has access and control over the data whether encrypted or un-encrypted as the

responsibility for the data maintenance (such as data backups and data restore) is usually mandated to them.

Although the cloud tenant outsourcing its IT functions enables them to focus on their core business processes, they also put their vital data resources at risk in the hands of the third party provider who may use it for their own gains. For example, selling the data to a competitor, or using it for other purposes other than has been agreed.

Cloud computing at the onset came with security challenges as a result of its resource pooling and multi-tenancy characteristics where multiple customers share the same resources, same application, same databases or in some cases same tables (Youssef and Alageel, 2012; Khatri et. al, 2013). As an example, a cloud provider computing resources may be pooled to serve multiple subscribers and this may put data at risk of getting into unauthorised hands through accidental or intentional disclosure. Thus, the CSP may accidentally or deliberately leak data or other vital resources to a competitor as they serve multiple subscribers (Khan and Yasiri, 2016; Shapland, 2017). A study by TriguerosPreciado (2013) found cloud computing security to be of a supreme concern to subscribers and this discovery in 2017 remains unchanged as confirmed by Ahmed (2017) study. The Treacherous 12 (2017) survey identifies data security breaches such as:

- The two Yahoo! data breaches reported in September and December 2016 (affecting 3 billion user accounts, leading to a drop of \$350 million in the acquisition price of Yahoo! which was earlier valued at \$4.8 billion) (McMillan and Knutson, 2017),
- Data loss such as malicious CSPs or malicious users intentionally corrupting the user's data inside the cloud by modifying or deleting (Chauhan, 2015; Sailaja and Usharani, 2017),
- Malicious insiders such as the theft of 1.5 million T-Mobile customers' data by an employee at their Czech offices (thehackernews, 2016),
- Denial-of-Service (DoS) attacks such as the Australian Bureau of Statistics denial of service (ABS, 2016) as concerns of cloud computing security.

Another issue that arises from the use of Cloud Storage as a Service is the use of customer data for marketing and personal profiling such as leaking it to competitors (Chauhan, 2015).

Ahmed (2017) study established cloud computing poses security threats to the subscriber in terms of:

- Who has access to the data/resource (accessibility)
- What other use is the data/resource been used for (usage)
- Where the data/resource is located (location)

- Who has ownership over the data/resources outsourced to the cloud (ownership)
- And also ensuring accuracy of the data outsourced for cloud storage (accuracy)

And these raise questions as follows:

- How can cloud data be secured to prevent unauthorised access?
- In what ways can a cloud subscriber prevent their data from being used for other purposes by the cloud provider?
- In what ways can the cloud subscriber ensure that their outsourced data is not vulnerable as a result of the data location since different countries have different data privacy laws?
- In what ways can the cloud subscriber ensure that they have sole ownership of their data outsourced for cloud storage?
- In what ways can the integrity of data outsourced for cloud storage be maintained?

In relation to ownership there is the risk in terms of what happens to the data on contract termination or in the event of conflict between the cloud subscriber and the cloud provider. For example, when a CSP refuses to grant a subscriber access to their data in the event of a dispute over say the subscriber's subscription payments.

With the issue of location, accessibility, and usage of the data resource, cloud computing distributes data across servers setup and managed by CSPs across the globe and this makes it difficult for the cloud subscriber to find in which country(s) their data is being stored, who has access to the data, and for what unauthorised use (Rao and Selvamani, 2015). Finally data outsourced for cloud storage can be altered in transmission by man-in-the-middle attack or modified inside cloud provider's storage facilities by a malicious insider attack (Sailaja and Usharani, 2017).

These issues are making it unattractive for organisations and individuals to subscribe to cloud services. Although traditional counter security measures such as using encryption techniques (for confidentiality), using hash functions (for integrity), and using firewall, anti-virus, intrusion detection and prevention systems (for availability) have been employed, they have been inadequate to securely protect vital organisation data against attacks. Malicious attackers have found ways of going round them to compromise vital business data asset using network security attacks as Dos/DDoS, U2R attack, R2U attack, Probing attack, MITM attack, Message replay attack, and Brute-Force analysis attack (thehackernews, 2017).

According to Wang (2009), cloud computing technology distributes data on multiple servers belonging to a single CSP but the challenge as noted by Ahmed (2017) is implementing a distributed protocol architecture that assures of a robust secured cloud data security in a defence-in-depth design.

1.3. Aim of Research

This research therefore seeks to propose a cloud data security solution framework and implement an algorithm whereby data outsourced for cloud storage will first be sliced into chunks of data fragments and then encrypted on the subscriber's gateway system before being distributed to multiple different CSP's storage nodes (storage servers).

1.4. Specific Research Objectives:

- i. To enhance security of data outsourced for cloud storage by ensuring the data is useful to only the data owner
- ii. To propose a cloud data security solution framework and an algorithm for securing cloud data that alleviates the cloud subscriber's fear of the data/resource's privacy, usage, location and ownership.
- iii. To implement and test the propose cloud data security solution framework and algorithm.

1.5. Research Questions

- i. How can one ensure data outsourced for cloud storage is useful only to the data owner?
- ii. Can a proposed cloud data security solution framework and algorithm alleviate the cloud subscriber's fear of the data/resource's privacy, usage, location and ownership?
- iii. How does the proposed algorithm compare in terms of strength and performance?

1.6. Justification of choice of the study

As outlined in the problem statement, cloud computing comes with numerous benefits but also faces several security issues especially in terms of data privacy, data integrity, and data availability. Cloud computing characteristics of resource pooling, multi-tenancy, on-demand self-service, broadnetwork access, and rapid elasticity introduces new security threats in terms of data accessibility, data ownership and data accuracy and hence demand new approaches for dealing with them.

Traditional counter security measures have been found to be inadequate for dealing with cloud security issues, an example been that encrypting data before sending it to a single CSP does not protect the data from been decrypted, deleted, or altered (O'Reilly, 2017).

A survey conducted by the Cloud Security Alliance CSA in 2016 identifies twelve security concerns of cloud computing including data breaches, data loss, malicious insiders and Denial of Service among others (The Treacherous 12, 2017). Other cloud security issues includes: the cloud provider profiting from using the subscriber's data entrusted in their care for advertising, or using the data to learn more about the subscriber for their own interest or gains. Although research suggests that cloud security threats from multi-tenancy architecture have been reduced by major CSPs such as Amazon and Microsoft, the threats are still real especially for smaller CSP's (Shapland, 2017).

In addition, although different countries have different privacy and security laws, acts, and regulations that govern the protection of data for example, the Asia Pacific Economic Cooperation (APEC) privacy framework, the Organisation for Economic Corporation and Development (OECD) privacy framework and the European Economic Area (EEA) data protection laws, the actual responsibility of ensuring that data and other resources outsourced to the cloud are secured and protected against data loss, damage, misuse usually rest with the custodian of the data - the CSP (CSA, 2011; OpenCirrus, 2017).

However, given that data is the life blood of every serious organisation and that with cloud computing the subscriber's vital data asset is to be outsourced to third party organisation, it is critical the cloud subscriber take key interest in ensuring the safety of their data been outsourced for cloud storage. Hence, this study is of the same view with Fahmida (2016) that with cloud computing, the data owner (cloud tenant) even bears paramount responsibility in ensuring security of its data than the custodian of the data. Especially where critical business data such as trade secrets, financial data, employee data, or health data are been transferred for cloud storage. This assertion is more critical because when it comes to cloud computing service provision there is a chain of inter-dependency of services provisioning and hence tracing data leakage(s) could be extremely difficult. This study seeks to propose secure data security architecture and system that ensures data is useful only to the owner.

1.7. Methodology

The study will be carried out following the design research methodology and will employ a plan-driven incremental development and delivery method in which the sub-systems and its components specifications are planned ahead but the design and development processes are carried out as a series of increments to deliver the system as increments. The software increments are programmed and provision to users for their rapid feedback. The subsystems are then integrated and tested to deliver the system. The proposed cloud security solution framework will be developed into software in an experimental lab-setup using JAVA, SQL, and PHP.

The major steps involved in the development process are outlined.

- i. To apply *erasure coding technique* to first sliced data objects outsourced for cloud storage into chunks of data fragments (Tashi and Ponsam, 2016; Plank, 2013).
- ii. To apply an *encryption algorithm* to encrypt the chunks of data fragment (Goswami and Singh, 2012).
- iii. To apply *data dispersion technique* to shuffle the encrypted data fragments and distribute to multiple CSP's storage nodes (servers) (CSA, 2011).
- iv. To ensure efficiency especially during data retrieval as different CSP's storage nodes host the data fragments and hence may be operating at different data rates, a *buffering technique* is used to buffer the data fragments from the fast storage nodes as a waiting mechanism until the data fragments from the delayed storage nodes are received and assembled for onward delivery to the subscriber.

By employing above measures, this study hope to address the issues raised in the problem statement and assure the cloud subscriber the security of their data as the encrypted data fragments will be of no value to the CSP.

- v. Finally the study foresees performance to be likely affected as security is strengthened and hence seeks to cater for performance by employing the use of *Metadata server* to keep track of the data fragments and where they are distributed so as to ensure accuracy of the cloud subscriber's data resources (Dell Power Solutions, 2005).

In effect, the study hopes the proposed data security solution framework and algorithm will assure cloud subscribers of the confidentiality, integrity and availability of their data resources outsourced for cloud storage.

1.8. The study organisation

The thesis is organised as follows:

Chapter 1: Introduction- gives brief introduction to the research, and then presents an overview of cloud computing technology, the statement of problem to be address, the overall study aim, the specific objectives, the research questions, and justification for the research.

Chapter 2: Literature Review- presents a review of related research work. The chapter reviews related published articles on the subject and also review related materials from other sources including Internet, Books, Journal Articles, among others.

Chapter 3: Methodology- presents the methodology employed for the study (Design Research), the software process method, and also presents the proposed model. A description of the proposed algorithm for securing cloud data on multiple CSP's storage nodes using Erasure coding, Encryption, Data Dispersion technique, Buffering technique, and Metadata works is presented.

Chapter 4: Implementation and Testing- implements the proposed data security framework and algorithm and test the result to ascertain the algorithm performance and most importantly its ability to assure cloud subscribers of the security of their data resource outsourced for cloud storage.

Chapter 5: Discussions and Results - present an insight into the results and compare the proposed system to other existing related systems in terms of its security strength.

Chapter 6: Conclusions and Recommendations- throw more focus on what the results actually means for both the cloud subscriber and the CSP especially in terms of benefits and suggest recommendations based on the research findings for the cloud subscriber and as well as suggest recommendations for future research work on the subject.

CHAPTER 2

LITERATURE REVIEW

2.0. The Concept of 'Cloud' Computing

The word cloud in the technology 'cloud computing' signifies the fact that the technology delivers computing from somewhere (i.e. a location in space) or a distance away from the user. Cloud computing technology disengages application and data resources away from the infrastructure and mechanisms that makes them available to the user (CSA, 2009). Thus, cloud computing is enabling

organisations to move away from viewing IT as a device-centric to a view that is application, data, and infrastructure centric.

Cloud computing is a new phenomenon that brings together many existing technologies and approaches to computing that is different from traditional computing setting which usually requires users to be in same location with their computing resources (Mell and Grance, 2011). The technology enables cloud tenants (individuals or organisations) to efficiently use existing computing resources such as software applications, software platforms, and hardware infrastructures across the world at a shared minimal cost. It also helps to expand the capacity of exiting computing resources smartly and cost effectively without having to re-design. It uses networked infrastructure software and hardware to provide computing resources to users in an on-demand environment.

Cloud services are often but not always utilised in conjunction with or enabled by virtualisation techniques. Thus, cloud computing services are to a large extent mostly based on virtualised resources (Kharche and Chouhan, 2012).

The cloud provider may own and house both the infrastructure and software required to run a home or business application for example as offered by Amazon Web Services – AWS. This is useful especially to small and medium enterprises (SME's) and even SaaS cloud providers as they often can't afford the up-front infrastructural capital needed to own the hardware and software required to setup their own traditional hosting environment that can effectively serve their customers. This can help SME's plan and assess their IT need as they only need to subscribe to a service required and can expand on-demand if needed without having to pay for redundant resources that will not ever be used. Thus, by subscribing to a cloud computing service that provides both hardware and software solution, the SME will not have to worry about forecasting its IT demands prior to actual usage which may lead to the problems of under-forecasting if the business flourishes or over-forecasting if the business deteriorates (Amazon Web Services, 2010).

Cloud computing service unlike traditional hosting service usually gives subscribers the opportunity to increase or decrease their resources utilisation capacity as and when needed in what is described as the rapid elasticity characteristic of cloud computing. All a tenant needs to have to use a cloud service is to setup an Internet enabled device such as computing devices (laptop, desktop, palmtop, and tablet) or a mobile device (PDAs, Smartphones, iPad, kindles) and an Internet connection via wired, wireless (WIFI), or mobile broadband.

Cloud computing gives end-users the benefits of accessing their computing resources from any location they may find themselves in the world provided they have a device with Internet connectivity, and the required resource providers systems are up and running. In addition, cloud computing can also help business to be more efficient and profitable as their staff can work 24/7 from anywhere (Huth and Cebula, 2011).

2.0.1. Characteristics of Cloud Computing

Cloud computing technology according to NIST has five (5) characteristics that distinguish it from traditional computing hosting as follows (Mell and Grance, 2011).

- i. *On-demand Self-Service*:- Cloud tenants or clients automatically request for and manage their own computing resources that suites their needs from a remote CSP without seeking any assistance from the provider.
- ii. *Broad Network Access*: - Means the service(s) is/are available over a network (whether private or public) and can be accessed via use of an Internet enabled device with Internet connectivity.
- iii. *Resource Pooling*:- Means cloud tenants or clients usually accesses resources (for e.g. storage facilities) that are pooled from different providers remote data centres and networks (virtual and physical) in a distributed computing model that are dynamically assigned and reassigned based on demands.
- iv. *Rapid Elasticity*: - Service providers can scale their services small or large rapidly (in some instances automatically) depending on resource availability or user demand. The cloud subscriber on the other hand presumes service availability to be abundant and hence can scale their resource utilisation capacities at any moment required.
- v. *Measured Service*: - Service usage is measured and controlled for the purpose of optimising usage of the available resources and also ensure fairness in usage. In some cases usage is metered and the consumer is billed according to their use (i.e. the pay-per usage model).

Figure 2.1 illustrates the overall concept of cloud computing.

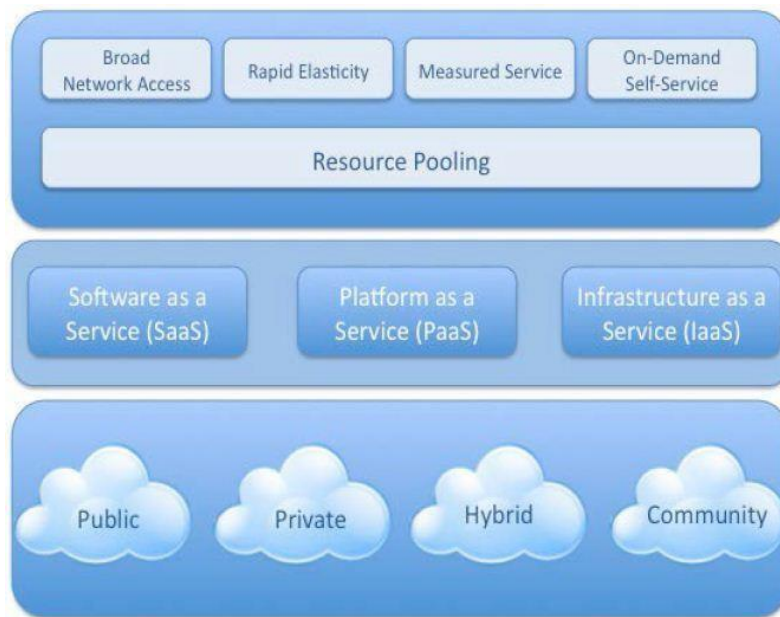


Figure 2.1 – NIST visual model of cloud computing (Mell and Grance, 2011)

The Cloud Security Alliance (CSA), in addition to the above five cloud computing characteristics from NIST also identified **Multi-Tenancy** as an important cloud feature. The CSA define *multitenancy* as “use of same resources or application by multiple consumers that may belong to the same organisation or different organisation”. Multi-tenancy enables virtualised resources to be pooled from systems belonging to same organisation or different organisations to serve multiple consumers as shown in Figure 2.2a and Figure 2.2b.

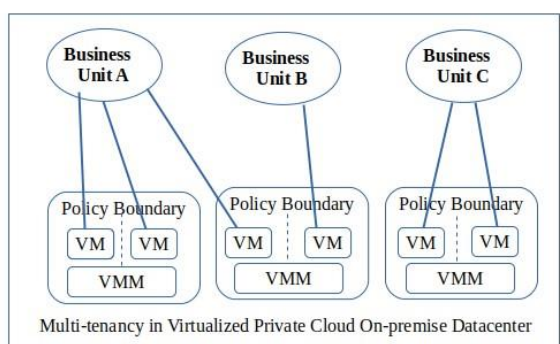


Figure 2.2a – Private Cloud of a company with 3 business Units (CSA, 2011)

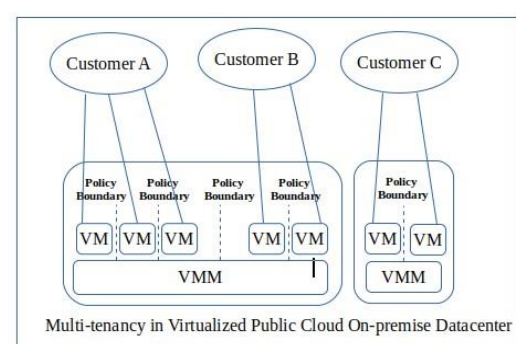


Figure 2.2b – Public Cloud provider with 3 business units (CSA, 2011)

To understand cloud security issues and other risks implications of outsourcing our data resources to the cloud, it is important we first analyse the various cloud computing service models and deployment models.

2.0.2. Cloud Service models

Cloud computing offers subscribers the option to subscribe to one of three service models referred to as 'SPI models' where the 'S' stand for Software-as-a-Service (SaaS), the 'P' for Platform-as-a-Service (PaaS), and the 'I' stand for Infrastructure-as-a-Service (IaaS). Each of the cloud service models is described with each serving a specific function that gives subscribers specific control. They differ in the levels of control and their service offering to the subscriber. Figure 2.3 illustrates the architecture of cloud computing service models.

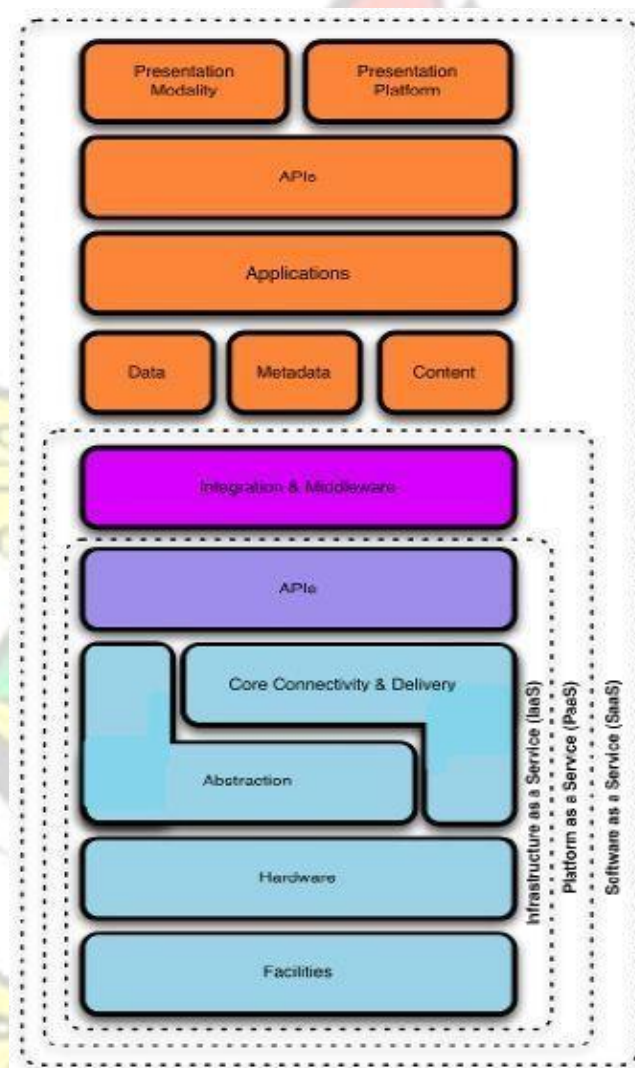


Figure 2.3- Architecture of cloud service models (CSA, 2011)

The IaaS model also known as Infrastructure Cloud is the foundation of all cloud services and provides subscribers access to IT infrastructures such as the networking infrastructures, storage system, servers, and the virtualised environment they need to run their software applications – Figure 2.4. Thus, an IaaS provider such as SAVVIS, Amazon, and Google respectively provides the hardware infrastructure,

virtualised software platform and network infrastructure needed to run the cloud subscribers application. The IaaS model enables IT infrastructures to be setup and used via remote access and made available to subscribers on an elastic basis.

Cloud Services

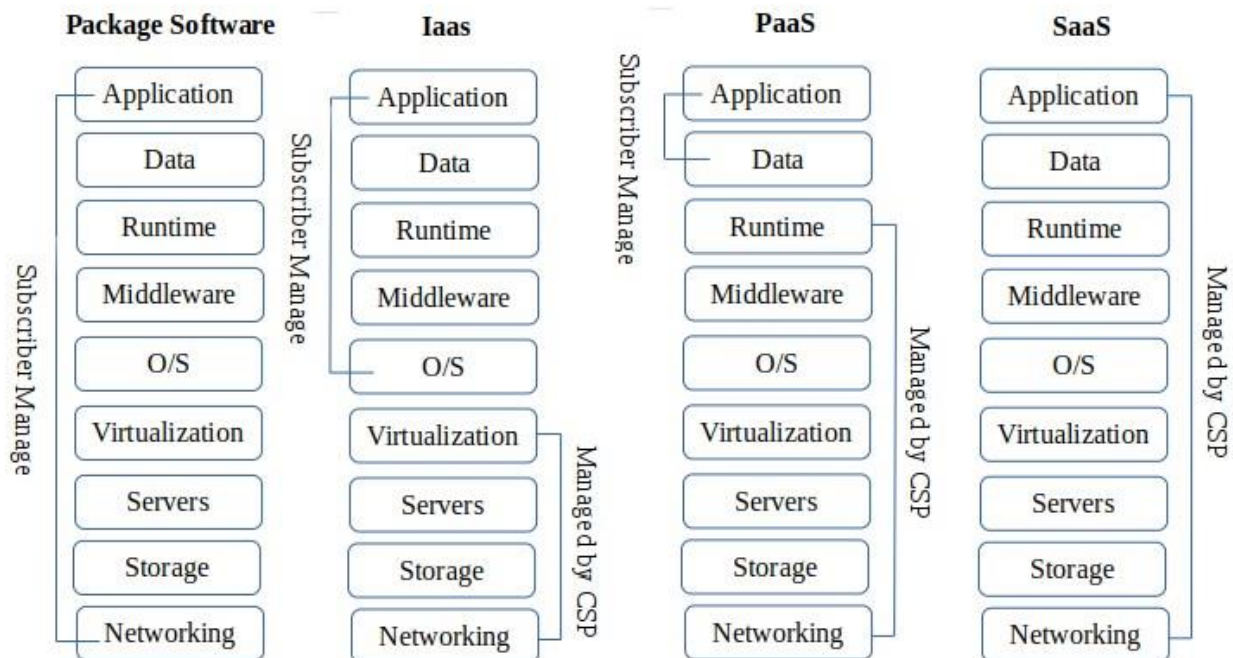


Figure 2.4 - Cloud services types and resource control (Satyanarayana, 2012)

The IaaS model requires that operating systems, applications, and other contents as middleware, runtime, and data outsourced to the cloud are controlled and secured by the cloud tenant (the client subscriber or the client provider) while the IaaS provider only takes charge of controlling and securing their cloud infrastructures. When a subscriber signs up to an IaaS agreement, they completely outsource their IT infrastructure needs to the total control and security of a third-party IaaS cloud provider (Satyanarayana, 2012).

2.0.2.1. The Platform-as-a-Service (PaaS) Cloud Service model

The PaaS model, also known as Development Cloud, builds upon the IaaS model and gives the PaaS subscriber more control over the IaaS infrastructure. A PaaS cloud provider gives tenants (client subscribers or client providers) access to IaaS infrastructure plus software development toolkit and a number of supporting tools they need to develop and operate their own customised SaaS software.

application for which they have full control over – Figure 2.4. Thus, the PaaS Cloud provider provides, control, and secure the operating system, hardware, and network infrastructure needed by the PaaS subscriber and offer the subscriber the development tools to develop, operate, and manage their own customised software application(s) they have control over. In other words a client subscribed PaaS runs over its subscribed IaaS platform.

PaaS is aimed at giving subscribers opportunity to build their own controlled applications on top of a given platform. The PaaS model offers a balance between the IaaS and SaaS models where control and security provision is respectively left with the IaaS cloud provider and SaaS cloud provider. Thus, the PaaS cloud provider solely provides security and control for the infrastructure and development platforms while the subscriber is responsible for controlling and ensuring security of the applications developed on the platform. Examples of PaaS include YouTube, Google Apps, and Facebook (Satyanarayana 2012; Sailaja and Usharani, 2017).

2.0.2.2. The Software-as-a-Service (SaaS) Cloud Service model

The SaaS model also known as Application or Information Cloud is built upon an IaaS model and a PaaS models to give subscribers access to a SaaS cloud providers pre-built application and other resources needed to run it. A SaaS subscriber is an end-user of a SaaS cloud providers completely packaged application along with the platform (networking infrastructures, storage system, servers, virtualisation system, operating system, middleware, and runtime environment). The SaaS application is made available for the subscriber usage via a web browser online on the SaaS providers cloud infrastructures on demand under the provider's control (Mell and Grance, 2011).

Although some SaaS cloud providers offer their clients the option to download, install, and run a copy of their packaged applications on their computing devices locally over which the client has control, these applications are typically run directly from the Internet via a web browser as a service and are controlled by the provider. SaaS also makes it easy to run the same software concurrently on all connected client devices at the same time on the cloud.

The SaaS model offers the least control over the cloud to subscribers in comparison with the IaaS and PaaS models (Huth and Cebula, 2011). Like the IaaS model, the SaaS cloud provider is responsible for the management, control, and security of the packaged application deployed for the SaaS clients usage.

The control, management, and security of the infrastructures though is left to the IaaS provider in situations where the SaaS cloud providers application is hosted on another third party IaaS cloud providers infrastructure such as provided by Amazon.

In a cloud architectural stack of figure 2.4, control of cloud resources by the provider is higher up the stack where the provider has maximum control over the infrastructure, platform, application, and client data and lower down the stack where the provider has minimum control over the infrastructure, platform, application, and client data. Thus, the lower down the stack the CSP stops, the more security the subscriber is tactically responsible for implementing and managing. In a SaaS model, security controls including service levels, privacy compliance and their limits are negotiated into an agreement signed between a SaaS cloud provider and the SaaS subscriber. The SaaS cloud provider is not only responsible for the underlying infrastructure security as in the case of IaaS cloud but also responsible for the security of applications and other clients' resources as data hosted (Sailaja and Usharani, 2017).

2.0.3. Cloud Deployment Models

The three cloud service models; IaaS, PaaS, and SaaS are deployed under four cloud deployment models that are also used to describe a cloud provider's service offering to a cloud subscriber. The deployment models are Public cloud, Private cloud, Community cloud, and Hybrid cloud (Figure 2.1). Regardless of which cloud service model a subscriber chooses at least one of the deployment models discussed below must be chosen.

2.0.3.1. Public Cloud Deployment Model

This cloud deployment model is deployed over the publicly accessed Internet. Public cloud can be accessed by anybody with an Internet enabled device and an Internet connection who chooses to sign-up and use the service. The entire infrastructure and sometimes software used to build and run this deployment model is owned and operated by the CSP.

Public cloud service providers usually run and provide support for software from different cloud computing tenants such as banks, educational institutions, insurance companies and the CSP's costs of setting up and running the infrastructure is shared by the service utilisation companies according to their respective resource usage (Kharche and Chouhan, 2012). Public cloud deployment model are usually targeted at the general public or large industry groups. For instance e-mail services such as yahoo mail, g-mail, hotmail, online data storage services such as dropbox, google drive, social

networking services such as Facebook, Twitter, LinkedIn, and many more including services from enterprise applications as some online banking services, are publicly deployed cloud services. Public cloud deployment model is almost always set-up away from users' premises. Security implications of this deployment model include:

- i. The CSP using subscriber's data for other purposes than has been agreed such as for advertising and marketing purposes
- ii. The CSP claiming ownership of the data. For example holding the data on contract termination such as noted in problem statement with Facebook

2.0.3.2. Private Cloud Deployment Model

Unlike public cloud that is owned and operated by the cloud provider, a private cloud is set-up by a service provider for the sole ownership, usage, and control of a contracted organisation. Although the CSP may own the infrastructure, its entire operation, management and control is left to the subscriber third party organisation. Private cloud is usually set-up for a specific group or organisation use. Access is limited only to that specific organisation or group (Huth and Cebula, 2011). Services offered on private cloud may vary depending on the contracted organisation need(s) for setting up the private cloud hosting environment. Private clouds just as public clouds pool resources between different systems within one organisation or pool resources from systems belonging to different organisations to serve a specific purpose. Private cloud may be set-up on or off a contracted organisations premises. Some security issues of this deployment model include:

- i. A malicious insider leaking subscribers data to competitors such as the theft of 1.5 million T-Mobile customers' data by an employee at their Czech offices (Wei, 2016)
- ii. Data loss such as malicious CSPs or malicious users intentionally corrupting the user's data inside the cloud by modifying or deleting (Chauhan, 2015; Sailaja and Usharani, 2017)
- iii. The CSP services not been available when users need the service. For example in the event the CSP been hit by a DoS/DDoS attack as Australian Bureau of Statistics denial of service (ABS, 2016)

2.0.3.3. Community Cloud Deployment Model

Community Cloud services are usually shared by several organisations with similar requirements. Access is limited only to those organisations and they may choose to control, manage, and support the cloud resources themselves or leave to the community CSP to do so. The infrastructure may be owned by the community CSP or the collaborative organisations may choose to pool resources together

physically or virtually to set-up and own the infrastructure which may be sited at a location chosen by the concerted organisations (OPC, 2011). Security vulnerabilities of both the Public and Private Cloud Deployment models are all applicable to the Community Cloud model. Except that the threat level is lower in comparison to the Public Cloud model but higher in comparison to the Private Cloud.

2.0.3.4. Hybrid cloud deployment model

This is a combination of at least two of the already discussed cloud deployment models that respectively maintain their unique characteristics but are bounded together by standardized technologies that permits data and application portability e.g. combining private cloud and public cloud (Kharche and Chouhan, 2012). This model is susceptible to the security challenges of both the Public and Private Cloud Deployment models.

Table 2.1 below gives a summary of the cloud deployment models discussed.

Table 2.1 - Summary of the cloud deployment models (CSA, 2011)

	Infrastructure Managed By ¹	Infrastructure Owned By ²	Infrastructure Located ³	Accessible and Controlled By ⁴
Public	Third party provider	Third party provider	Off-Premise	Untrusted
Private/ Community	Organization Or Third Party provider	Organization Or Third Party provider	On-Premise Off-premise	Trusted
Hybrid	<u>Both</u> Organization and third party provider	<u>Both</u> Organization and third party provider	<u>Both</u> On-Premise and Off-Premise	<u>Both</u> Trusted and Untrusted

The respective differences existing between the various cloud service models, cloud deployment models, for example infrastructural locations, infrastructural ownership, infrastructural accessibility, and cloud control among others means the cloud subscriber is faced with a challenge when making a decision to migrate resources to the cloud. This implies the subscriber is faced with an enormous responsibility when choosing a CSP. The cloud subscriber therefore ought to undertake thorough risks assessment and due diligence of resources been moved to the cloud and the cloud service provider. Unless a cloud provider is willing to disclose their security controls and extent to which they are implemented (which in our view is unlikely), the uncertainty surrounding security of resources entrusted to the care of CSP's by subscribers will remain. In other words until such a time that the cloud subscriber get to be acquainted with the security control mechanisms employed by

CSP's for securing resources entrusted in their care, the subscriber will continuously be faced with a dilemma when making a decision to migrate resources to the cloud (Khan and Yasiri, 2016).

2.1. Existing measures for securing cloud resources

Various techniques and solutions have been proposed for secure data on the cloud. The techniques all seek to address one or all of the CIA triad models. For a cloud tenant to sign-up to a cloud computing agreement there is a need for the tenant to evaluate its business practices to ascertain the security risks posed to the resource (Shapland, 2017).

2.1.1. Assessing risks and deciding on resources to outsource to the cloud

With cloud computing a subscriber chooses the resources they want to outsource to a service provider. The subscriber may choose to move only part of their resource; either data or application to the cloud or move all resources to the cloud. As an example, a cloud subscriber may outsource their application and data to the care of a chosen CSP but may retain control over some functions and services of its system. Alternatively the cloud subscriber may choose to host its application and data on its own facility and outsource only some functions or services to the cloud. The interdependencies of service offering from different parties that make cloud computing possible makes it very difficult for subscribers to know exactly where to assign blame when problem occur. This impasse results in less consumer confidence in cloud computing (CSA, 2009). Cloud subscribers therefore ought to be mindful of the risks of outsourcing a resource to a third party CSP. Subscribers must evaluate the intended resource for security risks using the Confidentiality, Integrity, and Availability (CIA) security triad model before subscribing to cloud service. All information security solutions according to TechTarget (2017) seek to address at least one of the CIA triad model.

2.1.1.1. Confidentiality

The confidentiality as a security attribute is primarily intended to assure that no unauthorised access to information is permitted and that accidental disclosure of sensitive information is not possible.

Confidentiality guards against both internal and external unauthorised access and seeks to render information intelligible only to authorised person. Confidentiality ensures secrecy of data and assures users of their data privacy (Tayseer and Amin, 2015). Confidentiality is achieved traditionally through using encryption techniques. Some other common standard traditional measures for ensuring confidentiality have included use of User IDs and Passwords, Smart cards, and Biometric data.

Computer network attacks such as User to Remote attack (U2R), Remote to User attack (R2U), man-in-the-middle (MITM) attack, or insider attack usually makes it difficult to deal with this security issue using traditional counter security approaches as listed (Higashi, 2014). Traditional techniques for achieving confidentiality have been found to be inadequate for dealing with cloud security issues (Ukil et. al. 2013; Khandelwal, 2017). And as a consequence resulted with data security breaches such as the two Yahoo! data breaches reported in September and December 2016 (affecting 3 billion user accounts, leading to a drop of \$350 million in the acquisition price of Yahoo! which was earlier valued at \$4.8 billion) (McMillan and Knutson, 2017). This study therefore seeks to propose using data dispersion technique also known as data split techniques which breaks data into fragments, encrypt them, and distribute them to storage nodes belonging to different CSP to achieve Confidentiality.

2.1.1.2. Integrity

The integrity as a security attribute ensures that data is clean and trustworthy by protecting systems data whether at time of entry, in transit, or in storage, against intentional or accidental alterations. Ensuring integrity has three goals as follows:

- Prevent unauthorised users from making modifications to data
- Prevent authorised users from making unauthorised modifications to data
- Maintain internal and external consistency of data

Integrity generally has been achieved through use of a hash function algorithm as MD5 or SHA1, and checksum however these standard measures have not sufficiently guard against tempering with integrity of data especially with the cloud (Tayseer and Amin, 2015; Khandelwal, 2017). Issues of data loss such as malicious CSPs or malicious users intentionally corrupting the user's data inside the cloud by modifying or deleting it have been reported (Chauhan, 2015). Also, malicious insider attack such as the theft of 1.5 million T-Mobile customers' data by an employee at their Czech offices is another data integrity issue of recent time (Wei, 2016). Message replay attack is mostly employed by attackers to achieve their intended purpose of altering the integrity of messages in transit. This study proposes to use data dispersion technique (aka data split techniques) to protect data integrity.

2.1.1.3. Availability

This promotes the accessibility of data for authorised use. In other words the availability as a security attribute seeks to make data and other resources outsourced to the cloud accessible to legitimately authorised subscribers. This security model traditionally has been achieved through use of countersecurity technologies as firewall, access control, file permission, system updates, redundancy

systems such as RAID, and etc. Perpetrators commits attacks such as Denial-of-Service (DoS) or Distributed Denial-of-Service (DDoS) to deny legitimate cloud subscribers access to resources hosted on their chosen CSP cloud infrastructures (Shapland, 2017). Other challenges faced by this model includes

- i. Loss of information system capabilities as a result of an occurrence of a natural disaster as (fire, flood, storm, earthquake, and etc) (Shearer, 2017), or human actions as industrial strikes, intermittent power failures, system maintenance, system upgrade, and etc.
- ii. Equipment malfunction or failure during normal use for example a server breakdown, internetwork device such as switches, routers, hubs, etc. malfunctioning.

Attack such as that of the Australian Bureau of Statistics (ABS) denial of service is a recent example of a DDos attack. The attack led to a shutdown of the ABS website (Lui, 2017). This study aims to thwart this security model using distributed parity technique and buffering technique.

Although cloud computing provides users with the advantage of Scalability (ability of offering unlimited processing and storage capacity, Reliability (ability of enabling user access to applications and documents anywhere at any-time where there is Internet enabled device and Internet connectivity, Efficiency (advantage of freeing-up organisations to focus on their core mandate to innovate and come out with new products, It also opens doors to serious risks to the Confidentiality, Integrity, and Availability of data as realised by the example security breaches above.

Cloud providers therefore ought to implement security mechanisms in overlapping layers to prevent, detect, and respond to unauthorised intrusion or unauthorised usage of resources to enhance subscriber confidence in cloud services. In the same manner, it is also vital cloud subscribers knows the security measures employed by their chosen CSP in combating the CIA security traits (Shah and Anandane, 2013). Cloud subscribers should undertake risk assessment of their chosen cloud provider by critically analysing their Incident management policies, Business continuity policies, Disaster recovery policies, Business processes and procedures, Location, Back-up facilities (Krishna et. al., 2016).

After assessments of the risks have been carried out, the cloud subscriber ought to determine their preferred cloud deployment model bearing the risks in mind. The subscriber must then select the cloud service model offering required and hence the type of CSP. The cloud service model required ultimately determines the level of control a subscriber relinquishes to a CSP. The subscribers focus

must be on negotiating for the level of control required from the CSP so that counter measures can be put in place to deal with the potential risks identified. These counter measures may include putting in place specific risk response strategies such as: Avoidance (meaning, moving away from activities that result with risks), Reduction (meaning, taking steps to minimise the risk or setting up mechanisms to reduce the risks), Share or Insure (meaning, insure against the risks or put in place strategy to share the risks), Accept (meaning, take no action against the risks due to cost involve or benefits to gain) (OPC, 2011).

As the cloud subscriber/tenant usually have no physical control over cloud infrastructure in most cloud setup, contract agreements, service level agreements (SLA's) and providers documentation become vital in managing risks than in a traditional enterprise owned hosting environment (Hussain et. al., 2017).

2.2. Cloud computing challenges

OpenCirrus (2017) identifies four major challenges of cloud computing as follows: Data Security and Privacy, Data Ownership, Lack of Standardisation, and Lack of resources and expertise. Out for the four, ensuring data security and privacy is noted as the biggest challenge today. This challenge was attributed to the fact that some CSP's may behave un-ethically by make money through using personal information of subscribers entrusted with them for advertisements and other purposes for which the data owner's permission has not be sought. Or the CSP may use the information to learn more about their subscribers for their own interest. In addition given that personal information may be transferred by the cloud subscribers CSP to another third party organisation (say a data centre) probably located in another country un-knowingly to the cloud subscriber, it is paramount to ensure that the information transferred is useful only to authorised persons. There is however the risk of the information falling into the hands of un-authorised persons or risks of the information being kept by the CSP or its allied partners for other purposes even when an agreement has been ended or annulled (Sailaja and Usharani, 2017).

Data ownership is seen as another major challenge of cloud computing. Different countries have privacy and security laws, acts, and regulations that govern the protection of data, for example, the Asia Pacific Economic Cooperation (APEC) privacy framework, the Organisation for Economic Corporation and Development (OECD) privacy framework, the European Economic Area (EEA) data protection laws, and etc., each of these laws according to a CSA (2011) report places on the custodian

of the data the burden of ensuring the protection and security of personal data. In cloud computing the data custodian is the cloud provider and most cloud contracts have clauses that make the custodian of the data the owner (OpenCirrus, 2017). This security challenge is a concern to subscribers and hence preventing widespread adoption of cloud computing.

Cloud computing technology presents new challenges to securing data and other resources usage than traditional IT hosting service. Cloud computing characteristics of multi-tenancy, resource pooling, rapid elasticity, on-demand self-service, and broad network access, require new data security approach. Although cloud computing comes with significant concerns about security, privacy, data integrity, intellectual properties, research suggest that cloud based service models provides better security to clients data and other resources than traditional IT models (OPC, 2011). This though is not as a result of use of superior counter security measures but because privacy and security laws as well as government acts and regulations compels cloud providers to put in place privacy protection and also use security mechanisms to secure subscribers data.

2.3. Summary of Cloud computing security issues

- CSP's who serve multiple tenants (individuals or organisation) may accidentally or deliberately leak data and other vital resources.
- Use of cloud data by CSP's to serve their own peculiar interest or for purposes other than that for which subscribers use their service(s) without seeking their consent.
- Privacy invasion by spoofing or spying on customers' data and other resources entrusted in their care.
- Data ownership on contract termination or abrogation of contract
- Distributed Denial-of-Service (DDoS) to deny legitimate cloud subscribers access to resources hosted on their chosen CSP cloud infrastructures
- Issues of data loss such as a malicious CSP or malicious insider intentionally corrupting the user's data inside the cloud by modifying or deleting

2.4. Some examples of Network Attacks on Cloud Computing

Network attacks come in many forms some common examples of attacks on cloud computing includes the following (Shah and Anandane, 2013; Alani, 2016):

2.4.1. DoS/DDoS Attacks

In this attack, the attacker (e.g. hacker) continually flood a network with false requests that keeps a server too busy and prevent it from responding to legitimate requests from authorised system users (e.g. clients) thereby denying them access to their needed service(s). Denial-of-Service (DoS) attack may be launched from a single computer or from a group of computers distributed on the Internet. The later attack is referred to as Distributed Denial-of-Service (DDoS). DoS attacks may occur as a result of undiscovered flaws in system implementation or other system vulnerabilities. For example a program developed by a developer who is unaware of a bug in the software that could crash the program if it encounter an exception such as unexpected user input or a program developed by a developer for which the developer intentionally insert a bug in the developed software to serve as a backdoor for later attacks of the system. Examples of DoS attacks include: Ping of death, Mail bomb, UDP storm, TCP SYN Flooding, and smurf. DoS attack remains the same in cloud computing just as in the traditional IT hosting model. However in Cloud Computing model DoS attacks get nasty as Cloud providers serves several multi-tenant customers and hence the impact of a DoS attack is far greater than in a traditional IT hosting model where a provider serves few clients. This is because with cloud computing, providers keep expanding their infrastructure resources with much faster and computational efficient systems to serve as many subscribers as possible. This makes the impact of DoS attack in the cloud environment enormous when it occurs because the cloud model makes available more computational power to the DoS attack. The impact of DoS attack on the cloud is even gargantuan when a DDoS attack is committed as more machines are compromised in the attack (InfoSec, 2017).

2.4.2. User to Remote Attacks (U2R), Remote to User attacks (R2U), and Probing attacks

In the U2R attack, the attacker log on to a system as a legitimate user with an authorised system user account and begin to exploit the systems vulnerabilities with the intent of gaining super user privileges to cause harm. This could be an insider e.g. an employee of the organisation. Examples of U2R attacks include: perl, xterm.

In the R2U attack, the attacker exploit a network by sending packets to a network system that he/she does not have legitimate access in order to access or expose its vulnerabilities. The attacker then exploits the privileges of authorised system users. Examples of these attacks include: xlock, guest, xnsnoop, phf, and sendmail dictionary.

With Probing attacks, the attacker continually scans a machine or a network to ascertain or expose its vulnerabilities so he/she can subsequently exploit it. This attack is used mostly in data mining. Examples of these attacks include: saint, portsweep, mscan, nmap.

These attacks may lead to a malicious insider (for example former employee, a contractor, or business partner) taking revenge by stealing data, modifying data, or deleting data (Higashi, 2014). Cloud subscribers who rely on one provider for service provision are at risk of this attack (Ahmed and Hossian, 2014; Fahmida, 2016).

2.4.3. Man-in-the-Middle attack (MITM), Message Replay attack, and Malware Injection attack

With MITM attack the attacker tries to comprise an existing network device (e.g. Router) or install his own router between two or more user devices. Using his installed device or the compromised device, the attacker can intercepts, modify, or fabricate data transmitted between user devices. The attacker then forwards the data intercepted as if they have not been tempered with. The attacker for example may

- i. intercept an IP Packet sent by USER 'A'
- ii. modifies its PAYLOAD, and
- iii. send the modified IP Packet to USER 'B' as if it comes from USER 'A'

By the attacker acting this way, both USER 'A' and USER 'B' believe they are directly talking to each other without realising the confidentiality and integrity of the IP Packets they receive have been compromised. For cloud computing model, if attacker succeeds in placing themselves between the cloud subscriber and the cloud provider then the MITM attacks will be possible and may lead to various forms of data security breaches such as data modifications, deletions, and may also lead the attacker to committing a message replay attack (Higashi, 2014).

In a message replay attack, the attacker first intercepts a legitimate message (for example as in MITM attack) keeps it intact, and then retransmits it at a later time to the original receiver.

With Malware Injection attack, the attacker focuses on injecting a service implementation (or an evil virtual machine) to a CSP's cloud environment with the main goal of taking full control of the subscriber's data hosted on the cloud (Chou, 2013; InfoSec, 2017).

2.5. An evaluation of standard counter-security measures used by CSP's

Examining available counter security measures employed by CSP's for dealing with computer network attacks such as those discussed above is one of the objectives of this study.

In other to properly secure data required that we take a look at the processes involve in the creation of data. Hence this study briefly outlines the data lifecycle.

2.5.1. Overview of the Data Lifecycle

Data today is accessed via different devices such as smartphones, laptops, PC's, ipads. The data generation process is a lifecycle that involve six stages or phases as follows (CSA, 2011):

- i. Create: - this stage is where a new digital data is generated or an existing digital data is modified, updated, or altered.
- ii. Store: - this is where the data created is put into storage for future use or reference.
- iii. Use: - the data created or data in storage is viewed or processed by the user.
- iv. Share: - the data is made available for sharing with other interested parties.
- v. Achieve: - the data is retrieved from active use and put into some sort of redundant permanent storage for future use.
- vi. Destroy: - the data is removed or deleted permanently from the system through use of physical or digital technique.

Although the above data lifecycle presents the stages involve in data generation, the CSA study did not touch on

- Where data is located (i.e. the place the data is stored)
- How data is accessed (i.e. who has access to the data), and
- Who has ownership of data

These issues are very crucial when dealing with the subject of data security especially with respect to cloud computing.

2.5.2. CSP's LOCATION as a data security issue

With respect to cloud data security, it is extremely important the cloud tenant (client subscriber or client provider) (e.g. a SaaS cloud provider hosting its application on IaaS cloud providers infrastructure) knows where their data resource(s) in cloud storage is located (Mahmood, 2011; Raisian and Yahaya, 2015). By getting to know the location, a subscriber for example may be able to check

the privacy and security laws, acts, and regulations that governs the protection of data in that country and know the extent of their enforcement (Sailaja and Usharani, 2017).

2.5.3. ACCESS as a data security issue

In addition to knowing the location of data in the cloud, it is important that the cloud subscriber knows who has access to the data and how the data is accessed in order to be assured of the data security (Rao and Selvamani, 2015; Ahmed, 2017).

2.5.4. OWNERSHIP as data security issues

After knowing the location of the data and also who has access, cloud subscribers must take keen interest in determining ownership of data outsourced for cloud storage. According to Gray (2014), ownership of cloud data depends on where the data was created. Thus, whether the data is created on the cloud provider's infrastructure or created on the cloud subscriber's system before upload. Service providers usually prevent access to their clients' data. For example, LinkedIn does not permit other services to access all of its user personal data such as the email address through their API. Also, Facebook's end-user-agreement states that the company stores data for as long as it is necessary and not as long as users want to keep their data. This implies users lose control and ownership of their data (FileCloud, 2016).

In summary, with cloud computing, data is distributed across servers setup and managed by CSPs across the globe and hence are subjected to different privacy and security laws, acts, and regulations. This distribution of servers across many countries makes it difficult for the cloud subscriber to find out the **location** of their data, or determine who has **access** to their data, for what unauthorised **usage**, and ultimately determine the data **ownership**.

2.6. Ensuring Data Security

Data security involves the use of specific measures, mechanisms, controls, and technologies to

- i. **Protect** unauthorised usage of data for example via using encryption techniques or through using Intrusion Prevention Systems - IPS
- ii. **Detect** unauthorised access to data for example through using Intrusion Detection System – IDS
- iii. Provide a **Response** to data security breaches as i and ii when it occurs. For example in the case of i, by using a much stronger encryption algorithm with bigger key size or by applying

other data security measures as the data dispersion technique that does not require encryption but instead distributes the data to multiple storage nodes of a provider. In respect of responding to ii, network firewall hardware or software may be installed.

In cloud computing, ensuring data security may involve the following three activities (CSA, 2011):

- i. **Detection and Prevention** of data migration to the cloud through using tools as

DAM – Data Activity Monitoring

FAM – File Activity Monitoring

URL filters – Universal Resource Locator filters, and

DLP – Data Loss Prevention

There are several instances of individuals in business units breaking corporate data policies and moving sensitive data to public cloud storage facilities as dropbox, google drive, box, etc., without the organisations approval (BBC, 2016). These tools help in detecting and preventing against these security breaches.

The operations and functionalities of each of these tools are outside the scope of this study. These tools are usually used in conjunction with other traditional data security techniques as encryption techniques and access control mechanisms to detect and prevent unauthorised migration of data to cloud storage facilities.

- ii. **Protecting** data in *transit* from a cloud subscriber to a cloud provider's infrastructures and also **protecting** data in *transit* between different cloud provider's infrastructures. This is traditionally achieved through encryption techniques however this study hope to address this issue with data split techniques based on erasure coding with encryption.
- iii. **Protecting** data once it is in the cloud provider's infrastructures. This is traditionally achieved through using access control mechanism as firewall, passwords, DMZ, and etc., but this study seeks to address this issue with data split techniques based on erasure coding with encryption.

2.6.1. Some current existing solutions for ensuring cloud data security

Rao and Selvamani (2015) propose that encrypting the data using the RSA cryptographic algorithm is the best solution to secure cloud resources. This claim is arguable as research shows encrypted files can be decrypted using a brute-force analysis attack. Also, O'Reilly (2017) note that hard drive based encryption are not safe and hence cloud subscribers ought to be mindful of how their CSP encrypt their

data. The CSP encrypting data on their server using software is much secure and recommended than using a drive-based encryption where the provider installs hard drives that automatically encrypt the cloud subscriber's data. Security researchers in 2015 for example uncovered flaws in a particular hard drive product line that enabled viewing encrypted data.

Another solution employed by CSP's has been to split data into pieces, encrypt the pieces and then distribute them to their distributed servers. Though this approach secures the data to some extent it did not protect data from being decrypted, deleted, or altered by a malicious insider. Hence a clientside encryption approach that gives access, management, and control of the encryption keys only to the cloud subscriber is employed to prevent data breaches caused by malicious insiders (Shah and Anandane, 2013; Chou, 2013). Client-Side encryption encrypts data at the subscriber's premises before the data is sent to the CSP. This solution although gives subscribers some level of assurance of their data security, O'Reilly, (2017) noted that encrypting data slices and sending them to a single CSP's storage facilities still poses a threat as the data slices can be re-assembled and decrypted, deleted, or modified by the CSP. Most CSP's including Google, Dropbox, BackBlaze B2, Box, Apache Hadoop, however uses this approach of splitting, encrypting and distributing data slices on their own storage facilities. This study therefore proposes a solution that distributes slices of encrypted data to multiple CSP's storage nodes and prevents a single CSP from having access to all of the data pieces.

CONCEPTUAL FRAMEWORK OF THE PROPOSED MODEL

The proposed system is designed to use encryption, Hashing, and Erasure Coding Technique based on Reed Solomon Coding and the Galois Theory. Hence, this section presents a discussion on them and how they are applied in this study.

2.7. Evaluation of Encryption Algorithms

For a perceptive operational critical data such as military or business financial data to be transmitted over an un-trusted public network such as the Internet, a system ought to be able to guarantee users of their privacy. Privacy also called confidentiality or secrecy has to do with making sure nosy people cannot read and make sense of a message intended for another recipient. Thus, the transmitted message should make sense to only the intended receiver. Privacy also means keeping the message out of the hands of unauthorised persons. In other words, privacy means ensuring an unauthorised person (an intruder) do not have access to the transmitted message in the first place.

Signals from microwave, satellite, and other wireless transmission cannot be protected from unauthorised reception. Even cable systems cannot always prevent unauthorised access because cables may be routed through out of the way areas such as basements or roof of buildings that may provide opportunities for malicious access and illegal interception of data.

Although it is unlikely any system can completely prevent unauthorised interception to transmission signal hence a more practical method that is traditionally employed for achieving privacy is to alter the message so only an authorised receiver can understand it. The method used to do this is termed **encryption** and **decryption** of information. By encrypting the message before it is transmitted the message is unintelligible to everyone that receives it except the rightful recipient. Thus, encryption means the sender of a message transforms the original message (called plaintext) to another unintelligible form (called ciphertext) and send the transformed unintelligible message out over the network such as the Internet to the intended receiver. On receiving the ciphertext the rightful receiver apply a reverse process of the encryption method used to re-transform the ciphertext back to its original form (the plaintext) in a process called decryption. The encryption/decryption methods process a message using an algorithm and a key. Thus, the sender uses an encryption algorithm and a key to encrypt and the receiver also uses a decryption algorithm (usually the reverse) of the encryption algorithm and a key to decrypt. Figure 2.5 illustrates the concept.

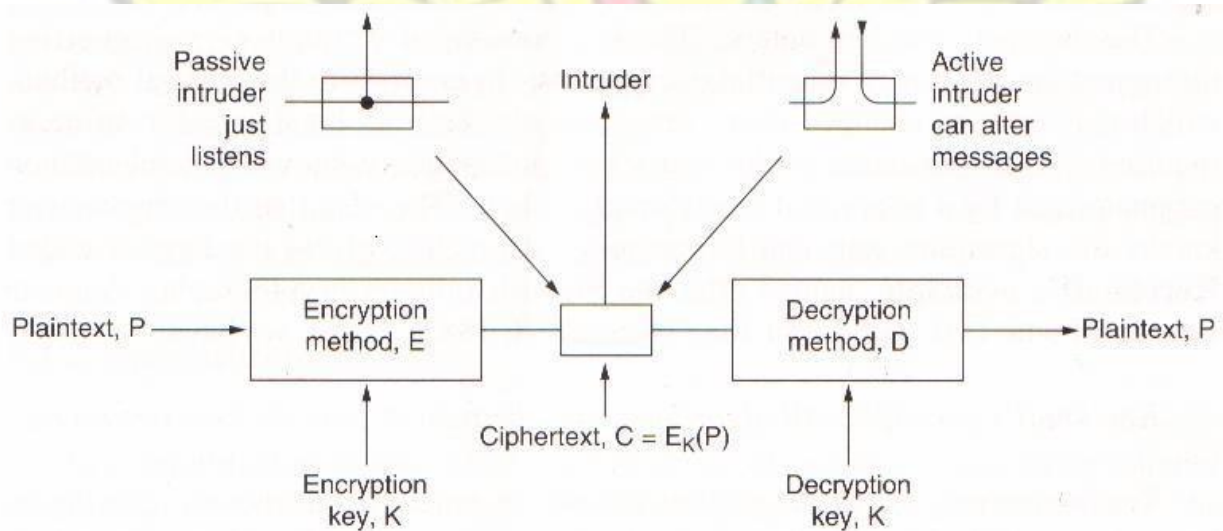


Figure 2.5 - Concept of encryption and decryption (Tanenbaum, 2003)

Encryption and Decryption methods fall into two broad categories as Conventional methods also known as Secret key methods or Symmetric methods and Public key methods also known as Asymmetric methods

2.7.1. Encryption and Decryption Methods

Many encryption methods exist and are usually classified according to the number of keys and the algorithm. In conventional methods, the encryption key and the decryption key are the same and the security of the method can only be assured if the shared key is kept secret between the communication parties. Also in conventional methods, the decryption algorithm is always the inverse of the encryption algorithm. For instance if an encryption algorithm is developed based on combination of addition and multiplication, the decryption algorithm will be developed based on a combination of subtraction and division. Hence anyone who knows the encryption algorithm and the key can deduce the decryption algorithm and therefore can decrypt an intercepted data whether in transmission or in storage. For this reason, security of conventional encryption methods can only be guaranteed only the encryption/decryption algorithm together with the key is kept secret.

Conventional encryption algorithms are broadly grouped into two, *character-level* ciphers or *bitlevel* ciphers. A cipher is a character-for-character or bit-for-bit transformation without regard to the linguistic structure of the message (Tanenbaum, 2003).

2.7.2. Character-level encryption methods

Encryption under character-level method is achieved via two techniques, *Substitution ciphers* or *Transposition ciphers*.

2.7.2.1. Substitution Ciphers

With Substitution Ciphers, each character in the plaintext is mapped to another character using an encryption key (k) value that indicates the corresponding ciphertext character that replaces it (Forouzan, 2001). This approach is referred to as *monoalphabetic substitution* encryption algorithm. As an example, suppose a sender is transmitting the word '**hello hello**' with $k = 3$. This will imply that each character of the plaintext would be replaced by a character located 3 positions ahead of it in the English alphabet (as is the case with *Caesar cipher* which shifts each character along by a number of places determined by value of k (Maiwald, 2003), resulting with '**khoor khoor**' as the ciphertext. Monoalphabetic substitution method is easily broken by snoopers as the alphabetic characters E, T, O, and A, frequent mostly in English words and hence a snooper can study the most frequent characters in a ciphertext and be able to determine the key. The algorithm maintains the order of the plaintext characters in the resulting ciphertext and hence an immature snooper can decrypt easily.

Polyalphabetic encryption algorithm is another type of substitution cipher. In this approach, the position of a character in the plaintext and also the character position in the English alphabet determine the corresponding ciphertext. Thus, the ciphertext is determined by shifting along the English alphabets according to the position value of the character in the plaintext. For example, the ciphertext **'igopt nltuy'** is transmitted for the plaintext **'hello hello'** when the polyalphabetic substitution encryption algorithm is used. Thus, character 'h' in position one of the plaintext is shifted by one position along the English alphabet to 'i', character 'e' in position two of the plaintext is shifted along by two positions to 'g', and so on. Though the polyalphabetic substitution algorithm is much difficult to decode, it does not also disguise the order of the character and hence it is still not secure. A skillful snooper will find it easy to undertake frequency analysis of the ciphertext in an attempt to decipher it as was the case with using the monoalphabetic substitution algorithm (Maiwald, 2003). **Vigenere cipher** is an example of polyalphabetic substitution encryption algorithm. In this cipher a keyword is used as the key to determine the character shifts in the plaintext to obtain the ciphertext. For e.g. using 'kafh' as the key on the plaintext **'hello hello'** gives the ciphertext as **'reqsy hjqsy'** which also can easily be broken.

2.7.2.2. Transposition Ciphers

With Transposition Ciphers, the characters in the plaintext are shuffled around instead of being substituted with other characters as in the case of substitution ciphers. Like substitution cipher, **transposition cipher** is another example of character-level encryption however in this technique; the plaintext characters keep their original form while their positions are altered to generate the ciphertext. The technique arranges the plaintext in a 2-dimensional table.

For example, the ciphertext **'PETHELTLDTSPLQEOOTEIRANUETGXSOCVAAX'** is transmitted for the plaintext **'Please do not touch Steve pet Alligator'**. The ciphertext is obtained through entering the characters of the plaintext into a table in row order where the table size determined by the number of columns is the encryption/decryption key and must be known to both the sender and the receiver (five in this example). The ciphertext is recorded vertically down the table from the first column while the plaintext is entered horizontally into the table - Figure 2.6 and Figure 2.7 (Tanenbaum, 2003).

P	L	E	A	S
E	D	O	N	O
T	T	O	U	C
H	S	T	E	V
E	P	E	T	A
L	L	I	G	A
T	O	R	X	X

Figure 2.6 - Transpositional Cipher

P	O	C	P	G
L	N	H	E	A
E	O	S	T	T
A	T	T	A	O
S	T	E	L	R
E	O	V	L	X
D	U	V	I	X

Figure 2.7 - Route Cipher

A transposition cipher is made more complex by specifying the key to determine the order of recording the columns for the ciphertext. For example the keyword 'KWAME' could be used to transform the plaintext above as entered into figure 2.6 to this ciphertext, 'EOOTEIRSOCVAAXPETHELTANUETGXLDTSPL'. The position of a character in the key and the order it appear in the English alphabet determines the order in which the columns are recorded to obtain the ciphertext. To decrypt, the key is used by the receiver to determine the number of table columns while the number of rows is determined by a count of the number of characters in the received ciphertext divided by the number of characters in the key. For instance, in the example above $35 \text{ ciphertext characters} / 5 \text{ key characters} = 7 \text{ rows}$. The ciphertext are then entered into the table following the order they appear in the English alphabet with their position in the key used in determining the column they are entered into. For example character 'A' in the key is considered first and as it is at position 3 in the key, the first seven characters of the ciphertext are entered into column 3. Likewise 'E' at position 5 of the key is treated next and hence the next 7 ciphertext characters are also entered into column 5 and so on. The plaintext is finally obtained by reading the characters from the rows. Using a key this way for a transposition cipher although makes it much harder for a snooper to decrypt, the approach is not that secure as the substitution cipher because the character frequencies are maintained and hence a more experienced snooper can decode through a trial and error attack or a frequency analysis attack although could be much difficult or a brute-force attack (Forouzan, 2001).

Some other well-known examples of transposition cipher include **Route Cipher**, which when used to encrypt the example message as entered into figure 2.7 would result with the ciphertext **XXROTAGPCOPLEASEDUVILLATEHNOTTOVETS** assuming the sender and the receiver

agree a key start point to be bottom right while routing up inward in anti-clockwise direction. This ciphertext can be decoded easily by choosing a route around the grid. Thus, the ciphertext is decrypted by entering the characters back into the grid using the key (comprising of the table size and the key start point). The plaintext is obtained by recording the text from the columns beginning from the first column.

Rail Fence cipher is another transposition cipher example. In this algorithm, the plaintext is written in a diagonal form into a grid of row of size determined by a key known to the sender and the receiver only. As an example, using the Rail Fence cipher to encrypt our example message with a key size of 5 indicating the number of rows in the grid as shown by Figure 2.8 gives the ciphertext below. The number of characters in the plaintext (35) determines the number of columns of the grid. The ciphertext is obtained by writing the text from a row at a time starting from the top row as follows: **PNSARLOOHTTLOEDTCEELTAETUVPIASOEG.**

P							N								S												A							R	
	L					O	O							H	T												T	L					O		
		E				D				T				C					E										E			L			T
			A		E						T		U							V		P									I		A		
				S								O										E										G			

Figure 2.8 - Rail Fence Cipher Encryption

The **decryption process** for Rail Fence cipher is carried out by first inserting the first character of the ciphertext into the upper left corner of the grid and then placing dashes (-) downwards this character diagonally. The dashes are then replaced with the other remaining characters of the ciphertext starting from the top row (Figure 2.9). Finally the ciphertext is decrypted by recording the characters starting from the top upper left corner of the grid diagonally in a zigzag manner to obtain back the transmitted plaintext.

2.7.4.1. Secret key encryption methods

Secret Key Algorithms also known as Conventional Encryption Algorithms or Symmetric Encryption Algorithms have five components as shown by figure 2.10

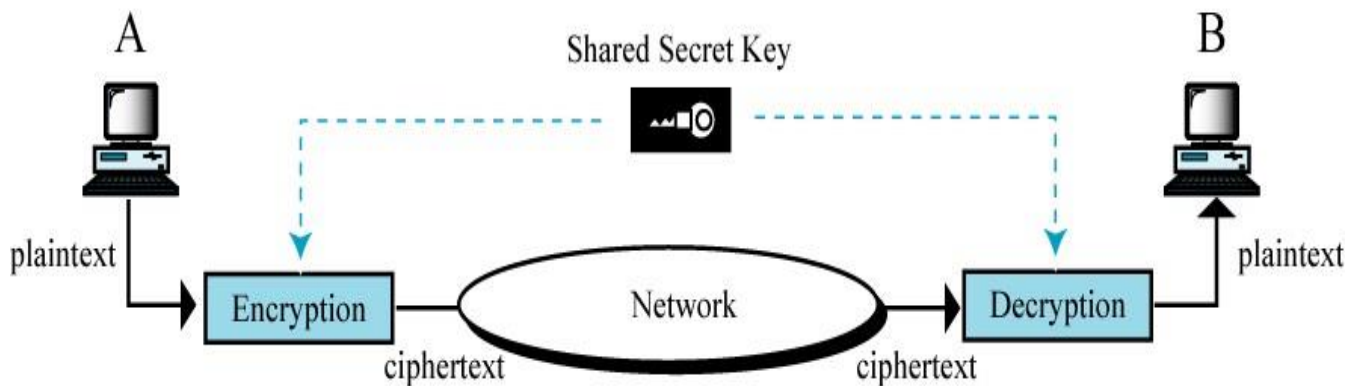


Figure 2.10 - Components of Conventional Encryption Algorithms (Stallings, 2003)

- Plaintext: This is the original message or data that is fed into the algorithm as input
- Encryption algorithm: This performs various substitutions and transformations on the plaintext
- Secret key: Also an input to the algorithm. The exact substitutions and transformations performed by the algorithm depend on the key
- Ciphertext: This is the scrambled message produced as output. It depends on the plaintext and the secret key. For a given message, two different keys will produce two different ciphertexts
- Decryption algorithm: This is essentially the encryption algorithm run in reverse. It takes the ciphertext and the same shared secret key and produces the original plaintext

There are two requirements for secure use of symmetric encryption:

- i. Both the sender and the receiver must ensure the secrecy of the shared secret key. The problem with meeting this requirement is that one of the communication parties generate the key and send to the other. The distribution of the key is a problem. If an attacker intercepts the key (MITM attacker) and gets to know of the algorithm, all communication using this key is readable.

- ii. For efficiency, there is a need for a strong algorithm. At a minimum, the algorithm ought to be such that an opponent who knows the algorithm and has access to one or more ciphertexts would be unable to decipher the ciphertext or figure out the key.

Conventional encryption methods are in wide spread use because the encryption and decryption algorithms are usually made public. The principal problem with Conventional Encryption methods is maintaining the secrecy of the key (Stallings, 2011).

There are numerous symmetric key encryption algorithms as follows: DES, 3DES, AES, IDEA, Blowfish, Twofish, Camellia, SAFER, KASUMI, SEED, Skipjack and RC5. These algorithms are generally categorized as being either stream ciphers or block ciphers (Kessler, 2017).

Stream cipher operate on a single bit, byte or computer word at a time and implement some form of feedback mechanism so that the key is constantly changing. A **Block cipher** is so-called because the scheme encrypts one block of data at a time using the same key on each block. In general, the same plaintext block will always encrypt to the same ciphertext when using the same key in a block cipher whereas the same plaintext will encrypt to different ciphertext in a stream cipher.

2.7.4.2. Public Key Encryption Methods

A significant development in the world of cryptography came in the 1970s by way of Public Key Cryptography (PKC). PKC allows two parties to engage in secure communication even when the channel is insecure, without having to share a secret key first. First described by Martin Hellman and Whitfield Diffie in 1976, PKC owes its strength to one-way functions - a mathematical function in which the inverses are significantly difficult to calculate although the functions themselves are quite easy to compute (Kessler, 2017).

By way of example, the multiplication of two integers such as 13 and 29 is a very easy computation and the result 377 can be obtained in a few milliseconds. However, factoring the result 377 of a multiplication to obtain the two factors 13 and 29 which produced it, takes a significantly longer time. Another example is the exponentiation operation, which takes a relatively little amount of time to process while taking the logarithm of the result in order to determine what two integers produced it, is a rather time-consuming operation. Such functions form the basis of Public Key Cryptography. The trick, though, is to discover a “trap door” in the one-way function so that the computation of its inverse becomes significantly easier when some item of information about the key is known.

Generic PKC makes use of two keys which are mathematically correlated. By knowing one key does not permit one to simply uncover its pair. Encryption of data is performed using one key while the other is used for decryption. It does not matter which key is used first, the second key is always able to decrypt whatever data the first key has encrypted. Due to the use of two distinct keys, PKC is referred to as Asymmetric Cryptography.

In Asymmetric Cryptography, each party in a communication has a private key which is known only to that party. Also, each party has a public key which is available for everyone in the communication. Any message that is transmitted during the communication can be encrypted using either the sender's private key (for Digital Signature) or the recipient's public key (for the purpose of achieving privacy).

If data is sent using the sender's private key, the recipient decrypts it using the sender's public key. The ability of the sender's public key to successfully decrypt the data gives authenticity to the message as the recipient is assured that the message was originally encrypted with the sender's secret key (thus, authenticating the sender). Similarly, it denies the sender the ability to deny having sent the message; since their public key was able to decrypt the message, it follows that the data was encrypted using the sender's private key. This is known as non-repudiation (Hansche et. al., 2013).

A number of Public Key Cryptography implementations have been developed since the concept was first described. Notable among them is RSA algorithm (named after Ronald Rivest, Adi Shamir and Leonard Adleman, who developed it). RSA is very widely used for key exchange, encryption of small blocks of data and digital signatures. The algorithm relies on the difficulty in factoring very large numbers and works by deriving a pair of keys from a very large number which is the product of two very large prime factors. The prime factors could be 100 or more digits long, giving rise to a product which is roughly twice the length of the prime factors. The significantly long number takes a very long time to factorize. A test in 2005 to factor a 200-digit number took 1.5 years and over 50 years of compute time (Kessler, 2017).

Other implementations of PKC are the Diffie-Hellman algorithm, Digital Signature Algorithm, ElGamal, Elliptic Curve Cryptography, Public Key Cryptography Standards, Cramer-Shoup cryptosystem, Key Exchange Algorithm, LUC and McEliece (Tanenbaum, 2003).

2.7.4.3. Hash Functions

Hash functions known also as Message Digests or One-way Encryption require no key. Instead a computation is carried out on the plaintext in such a manner that renders it difficult to recover either the contents or information about its length (Maiwald, 2003).

The values produced by hash functions can be used to provide a digital fingerprint of a file's contents so that when an alteration is made, the corruption of the file will be easily determined by the mismatch of newly computed hash and the previous one. Thus, one can verify that the copy of a file in their possession is authentic by simply computing its hash value and matching it against the one provided by the originator of the file. Hash functions prove very effective when detecting the activities of intruders and viruses (Stallings, 2011). Hash functions are also widely used to encrypt passwords.

There are several hash algorithms in common use presently. Among them are the Message Digest (MD) algorithms such as MD2, MD4 and MD5, and the Secure Hash Algorithms such as SHA-1, SHA-2 (comprising SHA-1, SHA-224, SHA-256, SHA-384 and SHA-512) and currently SHA-3. MD5 has been implemented in a large number of products even though several weaknesses in the algorithm were demonstrated by German cryptographer Hans Dobbertin in 1996 (Kessler, 2017). SHA-2, on the other hand has successfully withstood attacks.

Message Digest algorithms produce a 128-bit hash value from an arbitrary-length message. On the other hand, SHA-2 recommends using SHA-1, SHA-224 and SHA-256 for messages less than 2^{64} bits in length and SHA-384 and SHA-512 for messages less than 2^{128} bits in length (Kessler, 2017)..

Other known hash algorithms are RIPEMD, HAVAL (HAsH of VArIable Length), Whirlpool, Tiger and eD2k.

2.8. Reed Solomon Coding

In 1960, Irving Reed and Gastow Solomon proposed a system for encoding and decoding data that is quite versatile and efficient (Plank, 2013). Reed Solomon encoding works by adding special extra (parity) data (t) to the original data (k) being stored or transmitted. The extra data becomes handy when a portion of the transmitted or stored data (n) gets corrupted or lost and needs to be corrected or

regenerated ($n = k + t$). Reed Solomon recognizes two types of data corruption as follows (Plank, 2013):

Errors: With an error (data bits altering), the location of the corrupted or modified data is not known and must be computed as the roots of a polynomial refer to as error locator polynomial.

Reed Solomon coding can detect any number of corrupt data up to half the number of parity data.

Erasures: With erasures (data loss as a result of deletion), the location of the loss data is known. Hence there is no need to compute the locations. Reed Solomon coding can correct any number of erasures up to the number of parity data.

Reed Solomon (RS) limits the numbers it uses to a finite field (also called Galois Field) (cs.cmu.edu, 1998). A finite field is a set of numbers with rules such that all additions, subtractions, multiplications and divisions have results within the set. Hence powers, modulus and logarithms also have their results within the finite field. For use in programming, the preferred finite fields have a size that is a power of two (Benvenuto, 2012). This introduces some interesting properties for the finite field arithmetic. The denotation for such fields is $GF(2^m)$.

2.8.1. What is a Field?

A set of numbers is a field if it satisfies the following properties.

The real number system is primarily a set, for e.g. $\{a, b, c, \dots\}$, on which the operations of addition and multiplication are defined in such a way that for all pair of real numbers there exist a unique sum and product that are also real numbers and thus exhibit properties as follows (Trench, 2003):

2.8.1.1. Commutative Laws

$a + b = b + a$ ----- addition. E.g. $1 + 2 = 2 + 1$.

$ab = ba$ ----- multiplication. E.g. $1 * 2 = 2 * 1$.

2.8.1.2. Associative Laws

$(a + b) + c = a + (b + c)$ ----- addition. E.g. $(1 + 2) + 3 = 1 + (2 + 3)$.

$(ab)c = a(bc)$ ----- multiplication. E.g. $(1 * 2) * 3 = 1 * (2 * 3)$.

2.8.1.3. Distributive Laws

$$a(b + c) = ab + ac \text{ -----E.g. } 1(2 + 3) = (1*2) + (1*3)$$

There are distinct real numbers 0 and 1 such that

$$a + 0 = a \quad \text{and}$$

NB: For addition, the identity is '0'. $a * 1 = a$ Whereas multiplication identity is 1

For each 'a' there is a real number '-a' such that a

$$+ (-a) = 0$$

and if $a \neq 0$ there is a real number

$$\frac{1}{a} \text{ (or } a^{-1}) \text{ such that } a \left(\frac{1}{a}\right) = 1$$

$a + b \in \mathbb{R}$, and $a*b \in \mathbb{R}$ (closure laws),

Example

For a given set to be a field it should satisfy all the field properties.

An example of a field is the set of rational numbers.

$$\text{i.e. } a + b \in \mathbb{Q} \quad \text{and} \quad a*b \in \mathbb{Q}.$$

$$\text{e.g. if } a = 2 \quad a +$$

$$(-a) = 0 \quad \text{and} \quad a(-1) = 1$$

a

2.8.2. What is a Finite Field?

A finite field also known as Galois Field – GF is a field with finitely defined elements where upon performing the arithmetic operations of addition, subtraction, division, or multiplication of $f(p)$ mod p on any two of the field elements, the result is always an element in the set.

$$F_p = \{0, 1, 2, \dots, p-1\}$$

$\bullet, +$: integer addition and multiplication in modulo p

This property of a finite field enables its usage for error detection and data recovery in data communication and data storage (Wang, 2009).

A finite field is constructed using a **prime number base** or **powers of a prime number**. This is to ensure a unique value is obtained when addition and multiplication operations are performed on any two of the field elements. The elements of the finite field are the integers 0 through $2^m - 1$. Aside from 0, all the other field elements can be represented as a power of 2.

For example, finite field elements for GF(2) is constructed as $\{0, 1\}$, GF(3) as $\{0, 1, 2\}$, and GF(7) as $\{0, 1, 2, 3, 4, 5, 6\}$ (Wikiversity, 2016).

In the case of the powers of prime, a finite field for GF(2^n), where 2 is the prime base and ‘n’ is the exponent determines the number of elements in the field.

As an example GF(2^3) which is the same as GF(8) has field elements as $\{0, 1, 2, 3, 4, 5, 6, 7\}$, and GF(16) represented in prime powers of 2 as GF(2^4) has elements as $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15\}$. It follows therefore that GF(N) = GF(2^n) has field elements as $\{0, 1, 2, \dots, n-1\}$. Hence GF(2^8) = GF(256) has 256 field elements as $\{0, 1, 2, 3, \dots, 255\}$. This is an example of a modulus 256 field and hence 255 is the maximum value (Benvenuto, 2012).

For computer computational operations, a base 2 prime base is used for representing the field elements as prime powers (Wikiversity, 2016).

The elements of a finite field are usually represented as polynomials that take their coefficients from a particular field F_p . For example, for a polynomial, $F_p(x) = a_0 + a_1x + a_2x^2 + \dots + a_nx^n$ where $a_i \in F$. A deg.1 polynomial of $F_2(x) = a_0 + a_1x$, has field elements represented using alpha powers as $\{\alpha, 1+\alpha\}$ which are already in their irreducible form. Elements of deg.2 polynomials in $F_p(x)$ are $a_0 + a_1x + a_2x^2$ are obtained as $\{\alpha^2, \alpha^2 + \alpha, \alpha^2 + 1, \alpha^2 + \alpha + 1\}$ (Wikiversity, 2016).

Similarly for deg.3 polynomials in $F_p(x) = a_0 + a_1x + a_2x^2 + a_3x^3$ the elements are as follows $\{\alpha^3, \alpha^3 + \alpha^2, \alpha^3 + \alpha^2 + \alpha, \alpha^3 + 1, \alpha^3 + \alpha^2 + 1, \alpha^3 + \alpha + 1, \alpha^3 + \alpha, \alpha^3 + \alpha^2 + \alpha + 1\}$.

The application of Galois Field over the last few decades has been enormous especially in the areas of data communication and storage (Plank, 2013). Other applications have been as follows: encryption, and data compression.

Reed Solomon (RS) codes which operates over Galois Fields has been used extensively for the detection and correction of errors that occurs during data transmission and data storage (cs.cmu.edu, 1998). The study by Cox (2012) outline other application areas for RS codes as Voyager spacecraft, detecting and correcting data losses in wireless transmission, dealing with scratches on CD's, correcting scanning errors in QR codes among others .

2.8.3. Galois field elements construction

Galois field represented in binary form is very convenient for detecting and correcting errors in transmission and as well as for ciphering computer data. This is because it is a finite field and adheres to properties of a field. The elements of Galois field, $GF(P^m)$ is defined as $F_p^m = \{a_0 + a_1x + a_2x^2 + \dots + a_{m-1}x^{m-1}\}$ where $a_i \in F_p$.

$+$: addition in $F_p(x) \bmod p$, \bullet : multiplication in $F_p(x) \bmod \pi(x)$

Where $\pi(x) := \text{deg.} m$ irreducible polynomial in $F_p(x)$.

Irreducible polynomials (polynomial that cannot be factored) for example x^2+1 has no roots and are used to construct the elements of $GF(2^n)$. Reducible polynomials for example, x^2-1 , has roots as -1 and $+1$ and hence are not used when generating the elements of $GF(2^n)$. As an example, given $F_p = F_4 = F_2^2$, the elements of polynomials of $\text{deg} \leq m-1$ with coefficients from F_p are given as $F_p^m = F_2^2 = \{0, 1, \alpha, 1 + \alpha\}$, $p=2$, and $m=2$ (Lynson, No Date; Hill, 2013). For F_2^3 the field elements in powers of alpha are obtained as follows:

deg. 0	deg.1	deg. 2
a_0	$a_0 + a_1\alpha$	$a_0 + a_1\alpha + a_2\alpha^2$

$F_p^m = F_2^3 = \{0, 1, \alpha, \alpha+1, \alpha^2, \alpha^2 + \alpha, \alpha^2+1, \alpha^2+\alpha+1\}$, where $p=2$, and $m=3$.

A prime number is used to correct repetitions when the power of 2 exceeds the field size and needs to be wrapped to fit back within the field. An example is shown in Table 2.2 of the representation of the finite field $GF(2^3)$ as powers of 2, using the prime number 11 to handle or avoid repetitions.

Table 2.2 - Representation of finite field $GF(2^3)$ as powers of 2 using the prime number 11

Powers of 2	Expansion	Field element
2^0	2^0	1
2^1	$2^0 \times 2 = 1 \times 2$	2
2^2	$2^1 \times 2 = 2 \times 2$	4
2^3	$2^2 \times 2 = 4 \times 2 = 8 \equiv 8 \text{ XOR } 11$	3
2^4	$2^3 \times 2 = 3 \times 2$	6
2^5	$2^4 \times 2 = 6 \times 2 = 12 \equiv 12 \text{ XOR } 11$	7
2^6	$2^5 \times 2 = 7 \times 2 = 14 \equiv 14 \text{ XOR } 11$	5
2^7	$2^6 \times 2 = 5 \times 2 = 10 \equiv 10 \text{ XOR } 11$	1

For every field size, there is a set of prime numbers which can be used to uniquely identify the powers of 2 when they fall outside the field range (REDTITAN, 2011). The prime numbers used are usually found between 2^m and 2^{m+1} . In the case of the example above where the finite field is $GF(2^3)$, prime numbers within the range 2^3 to 2^4 can be used. The prime numbers that qualify are 11 and 13. Another example is shown in Table 2.3 using 13 to handle or avoid repetitions.

Table 2.3- Representation of finite field $GF(2^3)$ as powers of 2 using the prime number 13

Powers of 2	Expansion	Field element
2^0	2^0	1
2^1	$2^0 \times 2 = 1 \times 2$	2
2^2	$2^1 \times 2 = 2 \times 2$	4
2^3	$2^2 \times 2 = 4 \times 2 = 8 \equiv 8 \text{ XOR } 13$	5

2^4	$2^3 \times 2 = 5 \times 2 = 10 \equiv 10 \text{ XOR } 13$	7
2^5	$2^4 \times 2 = 7 \times 2 = 14 \equiv 14 \text{ XOR } 13$	3
2^6	$2^5 \times 2 = 3 \times 2 = 6$	6
2^7	$2^6 \times 2 = 6 \times 2 = 12 \equiv 12 \text{ XOR } 11$	1

As can be seen, using different prime numbers results in different assignments of powers of 2 to the field elements. However, all the elements are accounted for. Expressing the field elements as powers of 2 allows us to determine the logarithms of the field elements easily. For instance, if $2^5 = 3$, it follows that $\log(3) = 5$.

2.8.4. Galois Field (GF) Arithmetic

Arithmetic in GF or finite field is different from standard arithmetic. Unlike standard arithmetic, which has an infinite number of elements, there is limited number of elements in a finite field. Thus, arithmetic in finite field is basically carried out on a set of elements in which when the arithmetic operations of addition multiplication subtraction, or division is performed on the set the results is always found in the same set (Plank, 1997; Hill, 2013).

Recall, a finite field of elements P^n is basically represented in Galois field as $GF(P^n)$, where P is a prime base and n is the exponent of P , modulus P . For example, $GF(8) = GF(2^3)$ modulus 8 and the elements in the field are $\{0, 1, 2, 3, 4, 5, 6, 7\}$.

2.8.4.1. Addition and Subtraction in GF

Addition and subtraction in $GF(2)$ can be summed up as follows

+	0	1
0	0	1
1	1	0

−	0	1
0	0	1
1	1	0

XOR	0	1
0	0	1
1	1	0

Comparing with the XOR operation on bits, it is easy to see that both addition and subtraction boil down to an XOR operation on the field elements

The steps to performing addition and subtraction in GF(8) or GF(2³) are as follows:

- the polynomials of $\deg \leq m-1$ with coefficients from F_2^3 , where $m=3$ and $p=2$ is defined to obtain the field elements for F_2^3 as in section 2.8.3 above as:

$$\text{Elements of } F_2^3 = \{0, 1, \alpha, \alpha+1, \alpha^2, \alpha^2+\alpha, \alpha^2+1, \alpha^2+\alpha+1\}$$

- the addition table is constructed using the resulting elements of F_2^3 in modulus 2 as follows:

The GF addition table of Table 2.4 also has the entries for the GF subtraction operation as subtraction is performed as addition in computer systems using the additive inverse of the subtrahend. An element's additive inverse is the element that results with zero when added to the minuend. **Rule:** $a + (-a) = 0$

2.8.4.2. Multiplication and Division in GF

Multiplication in the finite field is easily performed using the logarithms. From the logarithm rule that

$$\log(a \times b) = \log(a) + \log(b)$$

A change of subject takes us to

$$a \times b = 2^{\log(a) + \log(b)}$$

The multiplication table for GF(8) mod 2 is constructed using the elements of F_2^3 and a deg.3 irreducible primitive polynomial in F_2^3 obtained as $\alpha^3+\alpha+1$ or $\alpha^3+\alpha^2+1$. This result with the field elements for F_2^3 as shown in powers of alpha and is used for the construction of the multiplication table, Table A1 (See Appendix 1).

$$F_2^3 = \{0, 1, \alpha, \alpha^2, \alpha^3, \alpha^4, \alpha^5, \alpha^6\}, \text{ mod } 2$$

Using the irreducible polynomial $\alpha^3+\alpha+1$ $\alpha^3 = \alpha + 1$ $\alpha^4 = \alpha(\alpha^3) = \alpha(\alpha + 1) = \alpha^2 + \alpha$

$= \alpha(\alpha^4) = \alpha(\alpha^2 + \alpha) = \alpha^3 + \alpha^2$, Now Substituting for $\alpha^3 = \alpha + 1$ gives $\alpha^5 = \alpha^2 + \alpha + 1$

$\alpha^6 = \alpha(\alpha^5) = \alpha(\alpha^2 + \alpha + 1) = \alpha^3 + \alpha^2 + \alpha$, Now Substituting for α^3 gives $(\alpha + 1 + \alpha^2 + \alpha) \text{ mod } 2 \quad \therefore$

$\alpha^6 = \alpha^2 + 1$ $\alpha^7 = \alpha(\alpha^6) = \alpha(\alpha^2 + 1) = \alpha^3 + \alpha$ Substituting for α^3 gives $(\alpha + 1 + \alpha) \text{ mod } 2 = 1$ $\alpha^8 = \alpha(\alpha^7) =$

$\alpha(1) = \alpha$, $\alpha^9 = \alpha^2$, $\alpha^{10} = \alpha^3$, $\alpha^{11} = \alpha^4$, $\alpha^{12} = \alpha^5$, $\alpha^{13} = \alpha^6$

As can be seen, the element values for alpha repeats from α^7 indicating GF(8) is a field.

Table A1 (See Appendix 1) also present entry values for division in GF(8) which is performed using multiplicative inverse of the elements in the set. **Rule: $a * (a^{-1}) = 1$**

As an example dividing α^5 by α^3 or (7/3) imply multiplying α^5 by the multiplication inverse of α^3 as follows: From Table A1, the multiplication inverse of α^3 is the corresponding element in the matrix that when multiplied by α^3 gives 1 as the result (i.e. **Rule: $a * (a^{-1}) = 1$**).

Hence the inverse of α^3 is $\alpha^4 = 6$. Therefore α^5 / α^3 is obtained ($\alpha^5 * \alpha^4 = \alpha^2$). Thus: 7/3 implies $7 * 6 = 4$ [where 6 is the inverse of 3].

Division in the finite field uses a similar logarithm rule as multiplication:

$$\log(a \div b) = \log(a) - \log(b)$$

Changing the subject gives

$$a \div b = 2^{\log(a) - \log(b)}$$

It can be seen from the Table 2.4 and Table 2.5 there are no identical entry in any of the rows or columns and there are also no repeating or negative entries in any row or column. These characteristics of the field element set makes the use of Galois field ideal for data recovery or error detection in data communication or data storage. Any of the elements in the set can be regenerated from the rest of the elements in the event of loss or damage. This is useful particularly in distributed data storage as cloud computing as in the event of a system breakdown or disk drives failures, the system can recover missing data and prevent any data loss. This system of data recovery is much efficient and cost effective than those of RAID technology (Plank, 1997).

Table 2.4 - Addition in GF(8) mod 2

+	0	1	α =2	$\alpha^3 \alpha + 1=3$	α^2 =4	$\alpha^4 \alpha^2 + \alpha =6$	$\alpha^6 \alpha^2 + 1=5$	$\alpha^5 \alpha^2 + \alpha + 1=7$
0	0	1	2	3	4	6	5	7
1	1	0	3	2	5	7	4	6

α $=2$	2	3	0	1	6	4	7	5
$\alpha^3 \alpha + 1 = 3$	3	2	1	0	7	5	6	4
α^2 $=4$	4	3	6	7	0	2	1	3
$\alpha^4 \alpha^2 + \alpha = 6$	6	7	4	5	2	0	3	1
$\alpha^6 \alpha^2 + 1 = 5$	5	4	7	6	1	3	0	2
α^5 $\alpha^2 + \alpha + 1 = 7$	7	6	5	4	3	1	2	0

Table 2.5 - Multiplication in GF(8) mod 2

*	0	α^7 $= 1$	α $= 2$	α^2 $= 4$	α^3 $= \alpha + 1 = 3$	α^4 $= \alpha^2 + \alpha = 6$	α^5 $= \alpha^2 + \alpha + 1 = 7$	α^6 $= \alpha^2 + 1 = 5$
0	0	0	0	0	0	0	0	0
α^7 $= 1$	0	$\alpha^7 = 1$	$\alpha = 2$	$\alpha^2 = 4$	$\alpha^3 = 3$	$\alpha^4 = 6$	$\alpha^5 = 7$	$\alpha^6 = 5$
α $= 2$	0	$\alpha = 2$	$\alpha^2 = 4$	$\alpha^3 = 3$	$\alpha^4 = 6$	$\alpha^5 = 7$	$\alpha^6 = 5$	$\alpha^7 = 1$

α^2 =4	0	$\alpha^2=4$	$\alpha^3=3$	$\alpha^4=6$	$\alpha^5=7$	$\alpha^6=5$	$\alpha^7 = 1$	$\alpha=2$
α^3 = $\alpha + 1$ =3	0	$\alpha^3=3$	$\alpha^4=6$	$\alpha^5=7$	$\alpha^6=5$	$\alpha^7 = 1$	$\alpha=2$	$\alpha^2=4$
α^4 = $\alpha^2 + \alpha$ =6	0	$\alpha^4=6$	$\alpha^5=7$	$\alpha^6=5$	$\alpha^7 = 1$	$\alpha=2$	$\alpha^2=4$	$\alpha^3=3$
α^5 = $\alpha^2 + \alpha + 1$ =7	0	$\alpha^5=7$	$\alpha^6=5$	$\alpha^7 = 1$	$\alpha=2$	$\alpha^2=4$	$\alpha^3=3$	$\alpha^4=6$
α^6 = $\alpha^2 + 1$ =5	0	$\alpha^6=5$	$\alpha^7 = 1$	$\alpha=2$	$\alpha^2=4$	$\alpha^3=3$	$\alpha^4=6$	$\alpha^5=7$

2.8.5. The RS Codeword

The ability to detect and correct data loss is of crucial importance to securing and recovering data stored on any storage facility (most importantly, the cloud). Reed-Solomon (RS) codeword is the most used for achieving this purpose. RS codeword is widely used for detecting and recovering data transmission errors. The following section illustrates how the RS codewords are generated and used for the detection and correction of errors in data transmission. According to Hill (2013) the RS codeword is generated using three (3) polynomials namely:

The “**Irreducible Polynomial**” (i.e. the polynomial equivalent of a prime number) is used as the *generating polynomial* for the Galois field elements generation. For the GF (8) elements generation, the irreducible polynomial ($\alpha^3+\alpha+1$) or 11 (i.e. 1011) is used.

The “**Generator Polynomial**” – This polynomial is required for generating the encoding polynomial (which is the 3rd polynomial needed for the generation of the RS codeword). The generator polynomial is a generic polynomial of the form;

$$G(x) = (x - \alpha) (x - \alpha^2) (x - \alpha^3) \dots (x - \alpha^{2t}),$$

where $\alpha^1, \alpha^2, \alpha^3$, etc. are the field elements and the value $2t$ determines the number of the Forward Error Correction (FEC) require. For example, assuming an RS codeword of **RS [7,5] s=3, t=2** where ‘s’ is the number of bits making a symbol size (in this case 3-Bit symbols), ‘t’ is the number of the 3Bits symbols used for error correction (in this case 2 (3-Bits) symbols), and 5 is the number of 3-Bits symbols used for representing the actual data chunks, while 7 is the total number of RS codewords for a GF (8). The generic expression (a.k.a. Maximum Distance Separable-MDS) for RS codeword is given as **RS [n, k] s, t**, where n is the number of codewords given as $2^s - 1$ and k is the number of data chunks.

Encoding Polynomial – The encoding polynomial which is the product of the polynomials of the form $(x - 2^i)$ for i values from 1 to the number of parity data is needed for the generation of the Reed Solomon (RS) codeword. For the RS[7,5]3,2 specification codeword example, 2 symbols are needed for FEC and hence only 2 of the Generator Polynomials are required as follows:

$G(x) = (x - \alpha^1) (x - \alpha^2)$. Since addition and subtraction operations give the same result in GF,

$G(x) = (x + \alpha^1) (x + \alpha^2)$. From Table 2.4, $\alpha = 2$, and $\alpha^2 = 4$ therefore $G(x) = (x + 2)(x + 4)$. Hence, $G(x) = x^2 + 4x + 2x + 8 = x^2 + 6x + 8$. 8 in binary is 1000 which is bigger than the largest field elements of 7 therefore the Generating polynomial of 1011 is XOR.

$$\begin{array}{r} 1000 \\ \text{XOR } 1011 \\ \hline 0011 = 3. \end{array}$$

Therefore, $G(x) = x^2 + 6x + 3$ is the encoding polynomial which is expressed also as **163** and is used for the RS codeword generation.

2.8.6. Reed Solomon Encoding

Reed Solomon coding treats data as polynomials and manipulates them as such. At the heart of this encoding technique is the production of a message polynomial $M(x)$ that is perfectly divisible by another predetermined polynomial (the encoding polynomial) $g(x)$ (Hill, 2013).

of data corruption. As an example, suppose transmission has respective ASCII representation as “49 50 51” and the CRC F (256) with parity of 2 given as $x^2 + 6x + 8$ or 16.

s given as

$$9x^5 + 50x^4 + 51x^3 + 52x^2 + 53x + 54$$

codeword for above message is to multiply the message polynomial by the generator polynomial

Forward Error Correction (FEC) code

$$2x^4 + 53x^3 + 54x^2 + 0x + 0$$

codeword for above message is to m

Forward Error Correction (FEC) code

$$2x^4 + 53x^3 + 54x^2 + 0x + 0$$

$$2x^4 + 53x^3 + 54x^2 + 0x + 0$$

$$2x^4 + 53x^3 + 54x^2 + 0x + 0$$

$6x + 8$ is used to divide into $M(x)$ as follows:



$M(x)$ above through polynomial long

ted by replacing the values in $M(x)$ with

$$M(x) = 49x^7 + 50x^6 + 51x^5 + 52x^4 + 53x^3 + 54x^2 + 186x + 244$$
 The

remainders are used for error detection and recovery.

2.8.7. Reed Solomon Decoding

Data transmitted is susceptible to corruption in various ways such as alteration or deletion. When an RS codeword is corrupted, there are a number of procedures that should be followed to recover the original data. For example, on receiving the codeword, the encoding polynomial is used to perform a polynomial division on it. RS codewords are designed to be perfectly divisible by an encoding polynomial (Plank, 2013; Twum et. al, 2016a). Thus, upon dividing a codeword by the encoding polynomial, a remainder of zero indicates that the data has not suffered corruption while any other remainder value means that the original codeword has been altered.

The detection of an error is invariably followed by an attempt to rectify it. It is possible to use remainder data as a means of determining the polynomial that represents the error so that it can be subtracted from the codeword to rectify the error. Some methods that have been used for correcting a corrupted codeword include:

2.8.7.1. Lookup Table

It is possible to use a lookup table (REDTITAN, 2011), to determine the error polynomial. This involves finding the remainders when all the possible errors are divided by the encoding polynomial. The remainders are recorded in a table that can be referenced when an error is detected. An example is shown by Table 2.6 using the encoding polynomial

$$x^2 + 6x + 3$$

on RS codeword of length 7.

Table 2.6 - Lookup Table Using RS Codeword of length 7

Index							
Value	x_0	x_1	x_2	x_3	x_4	x_5	x_6
1	1	$1x$	$6x + 3$	$1x + 1$	$7x + 3$	$7x + 2$	$6x + 2$

2	2	$2x$	$7x + 6$	$2x + 2$	$5x + 6$	$5x + 4$	$7x + 4$
3	3	$3x$	$1x + 5$	$3x + 3$	$2x + 5$	$2x + 6$	$1x + 6$
4	4	$4x$	$5x + 7$	$4x + 4$	$1x + 7$	$1x + 3$	$5x + 3$
5	5	$5x$	$3x + 4$	$5x + 5$	$6x + 4$	$6x + 1$	$3x + 1$
6	6	$6x$	$2x + 1$	$6x + 6$	$4x + 1$	$4x + 7$	$2x + 7$
7	7	$7x$	$4x + 2$	$7x + 7$	$3x + 2$	$3x + 5$	$4x + 5$

From Table 2.6, should a division of the codeword by the encoding polynomial yield a remainder of $6x + 4$, it can be surmised that the error has a value of 5 and an index of 4 (thus, $5x^4$). Similarly, a remainder of $4x + 4$ indicates the presence of an error with value of 4 at index of 3 (i.e. $4x^3$).

2.8.7.2. Advanced Error Correction

The lookup table works well when the RS codeword is short and the size of the finite field that the codeword uses is relatively small. Longer codewords, however, require much larger and more complex lookup tables. For longer codewords, a more advanced procedure is recommended for quickly and efficiently acquiring the error polynomial. The steps involved in the advanced error detection are as follows (cs.cmu.edu, 1998):

- Creation of a syndrome polynomial
- Solving the key equation to obtain the error locator and magnitude polynomials
- Searching for the values of the error locations
- Determining the magnitudes of the errors

The Key Equation and the Syndrome Polynomial - The positions of the errors in the received codeword can be thought of as the roots of a polynomial that is called the locator polynomial. From this perspective, it is necessary to find the polynomial which describes the locations of the errors in the

received codeword. Aside the locations of the errors, it is also necessary to know the value (or magnitude) of the error at that particular location. Another polynomial, called the magnitude polynomial, needs to be found which will evaluate at a specified location to give the magnitude of the error at that particular location.

Any error that is introduced into the Reed-Solomon codeword can be thought of as being comprised of specific additions of value at particular locations in the original codeword. As such, determining the error-locator and error-magnitude polynomials is key to discovering the errors in the codeword and fixing the errors.

If $C(x)$ denotes the original codeword, $R(x)$ denotes the received codeword and $E(x)$ denotes the error that was introduced into the codeword, then the following relations describe the relationship between the original codeword, received codeword and the error. The $R(x)$ and $C(x)$ will be given by eq. 01 and eq. 02 respectively as:

$$R(x) = C(x) + E(x) \dots \dots \dots eq. 01$$

$$C(x) = R(x) - E(x) \dots \dots \dots eq. 02$$

Since the codeword was created to be a perfect multiple of the encoding polynomial (Plank, 2013; Twum et. al, 2016a), evaluating the received codeword at the roots of the encoding polynomial will cause $C(x)$ in eq. 01 to become 0. As such evaluations that led to the production of the syndrome polynomial that are non-zero are in fact, evaluations of $E(x)$ at the roots of the encoding polynomial (Trench, 2003). In other words, the syndrome polynomial quantifies the error (REDTITAN, 2011). Attempts to solving the key equation to extract the error-locator and error-magnitude polynomials have the syndrome polynomial as the starting point. After the syndrome polynomial is generated, the next step is to determine the error locator polynomial and the error magnitude polynomial for that specific syndrome. Numerous algorithms have been proposed for achieving this. Some of which are the Peterson-Gorenstein-Zierler, the Berlekamp-Massey, and Sugiyama (Cox, 2012). However, two most widely used methods are the

- i. Berlekamp-Massey Algorithm
- ii. Euclid-Sugiyama Algorithm (or Extended Euclidean Algorithm)

Berlekamp-Massey algorithm - The Berlekamp-Massey algorithm, which locates the Linear Feedback Shift Register for a given binary output sequence, works by initializing the locator

polynomial to 1 then testing to see if the assumed locator polynomial is able to generate a portion of the syndrome (Cox, 2012). If it is able to generate the required portion, the next iteration checks to see if it is able to generate a larger portion of the syndrome. If it is unable to generate the required portion of the syndrome, the assumed polynomial is modified to generate the required portion and another iteration of the algorithm is ran. The iterations run till a locator polynomial is found which is able to generate the entire syndrome.

Euclid-Sugiyama Algorithm (or Extended Euclidean Algorithm) - The Euclidean algorithm is an algorithm that finds the Greatest Common Divisor (GCD) of two numbers. The same algorithm can be used to find the GCD of two polynomials (REDTITAN, 2011; Cox, 2012). The Extended Euclidean Algorithm was adapted by Sugiyama to solve the key equation, giving the error-locator and error-magnitude polynomials. The Euclid-Sugiyama algorithm works by first dividing x^n by the syndrome polynomial, then dividing the syndrome polynomial by the remainder from the first division, and then repeating the division-by-remainder process until a remainder is found whose degree is less than n , where n is the degree of the field polynomial. 2

The Berlekamp-Massey Algorithm is reported to have better and more efficient hardware implementation but the Euclid-Sugiyama Algorithm is adopted by this study as it is easy to implement in software (Cox, 2012).

Locating the error position using Chein Search Algorithm - Algorithm as Chein Search algorithm replaces the value of 'x' in the error locator polynomial (obtained from the utilisation of a Reed Solomon decoding algorithm such as Euclidean algorithm or Berlekamp Messey algorithm), with the inverse values of the Galois Field (GF) elements and evaluating using GF arithmetic (Wang, 2009; Twum et. al, 2016b). A computed value of zero (0) points to the location of the error (REDTITAN, 2011).

Correcting the error using Forney algorithm - The Forney algorithm which is of the form e_i

$$= x_i [\Omega(X^{-1}) / \Lambda(X^{-1})]$$

Where e_i is the value of the errored bit, x_i is the value of the GF element that pointed to the location of the error by the Chein search algorithm, $\Omega(x^{-1})$ is the magnitude polynomial, and $\Lambda(x^{-1})$ is the locator polynomial (REDTITAN, 2011).

2.9. Review of Existing Related Cloud File Storage Systems

2.9.1. Google File Systems (GFS)

Google is a major player in the world-wide web, generating exabytes of data from all the services they render such as the Google Search Engine, Gmail, Google Drive, Google+, Hangouts, Android etc. (Carson, 2016).

Instead of the traditional data centre, Google opts for a Distributed Computing System using large clusters of relatively inexpensive machines running on Linux operating systems. “Cheap” machines have a higher tendency to fail but Google’s ingenious use of their proprietary data storage system, Google File System (GFS), ensures that data is always available and none goes missing. According to Strickland (2017) in an official GFS report, Google revealed the specifications of the equipment it used to run some benchmarking tests on GFS performance. The test equipment included one master server, two master replicas, 16 clients and 16 chunk servers. All of them used the same hardware with the same specifications, and they all ran on Linux operating systems. Each had dual 1.4 gigahertz Pentium III processors, 2 GB of memory and two 80 GB hard drives. In comparison, several vendors currently offer consumer PCs that are more than twice as powerful as the servers Google used in its tests. Google developers proved that the GFS could work efficiently using modest equipment.

The GFS is designed to organize and manipulate huge files and to allow application developers the research and development resources they require. While some of the features of the GFS have been kept secret by Google, enough information has been released by the internet giant on how the GFS is structured as well as how it operates.

Being proprietary, GFS is optimized for the operations of Google and their clients. One such peculiarity is the optimization for appending data instead of overwriting it. This is because Google rarely needs to overwrite files; instead they add data to the end of files.

Much of the administrative duties required to keep the system running are automated, following after the principle of autonomic computing, a concept in which computers are able to diagnose problems and solve them in real time without the need for human intervention. In order to support autonomic computing on a large network of computers, the developers of the GFS designed it to be very simple. Users were given access to some very basic file commands such as open, create, read, write and close, with the addition of ‘append’ and ‘snapshot’ to meet Google’s special needs. ‘Append’ allows client computers to add information to an existing file without overwriting the previously written data while ‘snapshot’ creates a copy of the computer’s contents.

2.9.1.1. File Storage

Files on the Google File System are often very large – several gigabytes large. Moving files that large in a network uses up a lot of bandwidth. GFS resolves that issue by splitting each file into chunks of 64 megabytes (MB) each (Techopedia, 2017).

Each chunk is given a unique 64-bit identification number called a chunk handle. Keeping the chunks small helps with the bandwidth as well as making it easier to port chunks from one machine to another in order to balance workload across the system.

2.9.1.2. System Architecture

Figure 2.11 and Figure 2.12 shows the GFS architecture.

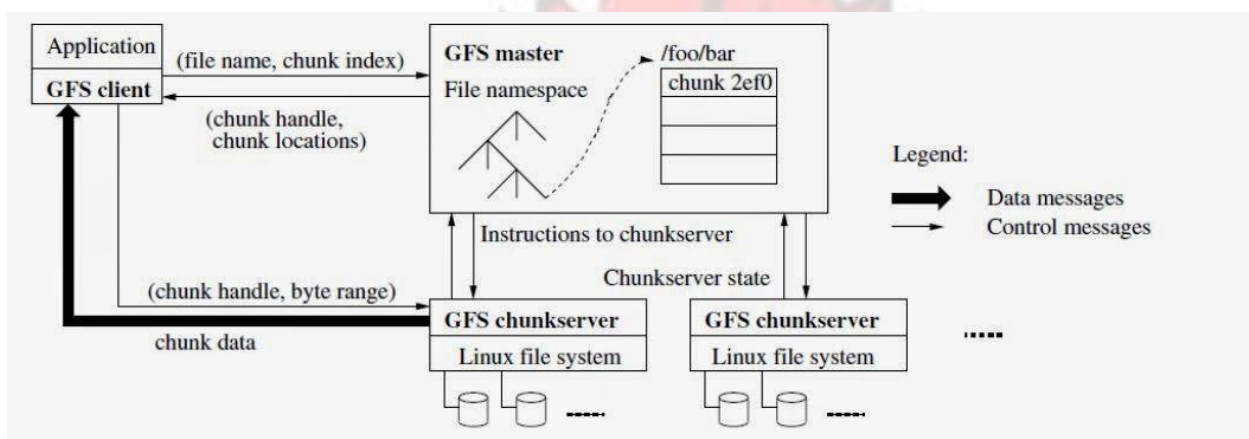


Figure 2.11 – GFS Architecture (Roshoan, 2017)

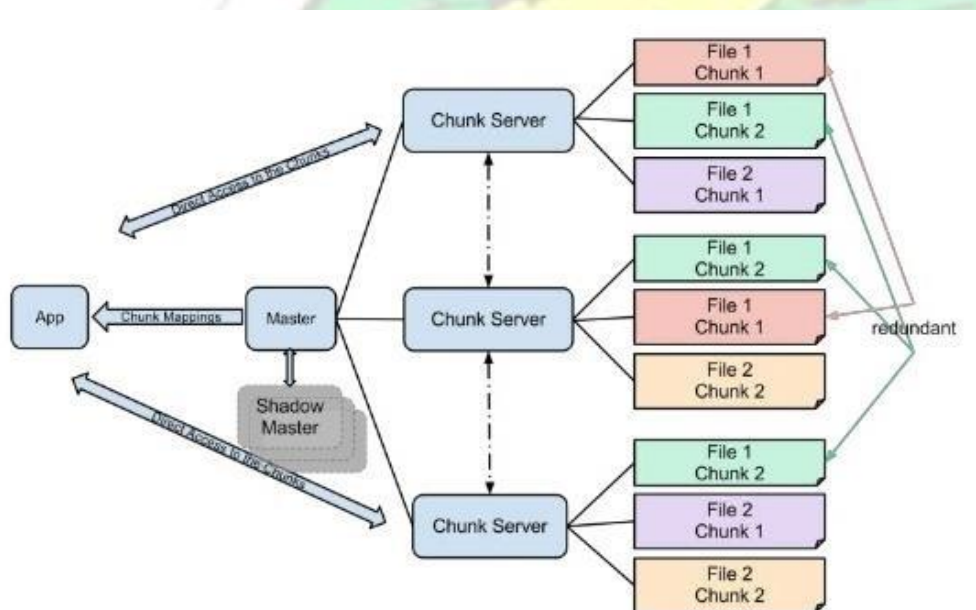


Figure 2.12 – GFS Architecture (Jain, 2013)

The GFS is organized into clusters of computers. Each cluster is a network consisting of hundreds or even thousands of machines. Each cluster comprises of a single master server and several chunk servers and clients - Figure 2.12.

Clients are any computers or computer applications which make a file request from the cluster. A client may make one of several requests including retrieving and manipulating existing files to creating new files on the system.

The master server coordinates activities and processes within the cluster. It maintains an operation log which tracks all the activities in the cluster. The master server also keeps track of metadata on all the chunks residing in the cluster. The metadata contains information on where the chunks reside in the cluster as well as how the chunks come together to form the full file. At start-up, the master receives data from all the chunk servers, telling the master the contents of their inventories. Every cluster has exactly one master server active at any point in time. However, there are secondary masters which keep copies of the master's contents. In the event where a master server goes down and fails to reboot, a secondary master server assumes the job of master server in that cluster.

Chunk servers do most of the work in a GFS cluster. The 64-MB file chunks are stored within the chunk servers. The GFS makes copies or replicas of the chunks and stores them on chunk servers in different locations or racks. This protects against data loss when a chunk server or rack of servers become inaccessible. The replicas are also identified as primary and secondary, with one primary and a backup of one or more secondary replicas. Chunk servers do not relay their file chunks through the master server when a client requests it. Instead, the chunk server connects directly to the client server and delivers directly to the client. This prevents a bottle-neck at the master server.

2.9.1.3. Read Requests (Download)

Read requests are handled simply. A client makes a request to the master server for the location of a particular file in the cluster. The master responds with the location of the primary replica of the specified chunk. After that the client contacts the chunk server directly, which in turn sends the replica to the client.

2.9.1.4. Write Request (Update)

Again, the client sends a request to the master server which replies with the locations of the primary and secondary replicas. The client sends the write data to all the chunk servers, starting the chunk server that's closest then continuing to the furthest chunk server. Upon receiving the data, the chunk

server with the primary replica assigns serial numbers to the changes to the file. The changes are called mutations. The order of the serial numbers instructs each chunk server how to apply the mutations. After assigning the serial numbers, the chunk server with the primary replica begins to apply the mutations to its own data, after which it sends a write request to the secondary replicas for them to imitate the action. When all the chunk servers have successfully updated their replicas, the primary chunk server notifies the client of the update success. In the event where a secondary server is unable to apply its updates, the primary replica instructs the secondary one to start the process from the beginning.

2.9.1.5. Erasure Protection

Google File System guards against data loss by making copies of the data chunks and storing the copies on different nodes within the cluster.

2.9.1.6. Data Access

The Google File System is proprietary and used by Google to manage their large data sets. In this case, the data is accessed solely by the owner since the data owner is also the owner of the storage units.

2.9.1.7. Use of the data/resource

The data stored in the Google File System is used solely by Google to cater for their application's data and processing needs. All uses of the data are therefore known and sanctioned by the owner.

2.9.1.8. Data/Resource Location

The data is stored in Google's clusters of commodity hardware. The equipment is owned by the data owners. As such the data is located in a known place.

2.9.1.9. Ownership

There is no contention as to the ownership of the data in the Google File System since the data owner is also the storage service provider.

2.9.2. Apache Hadoop

According to Apache Hadoop (2013), Hadoop is a collection of open source libraries for processing large data sets across thousands of computers in clusters. Hadoop consists of multiple libraries which together perform massive data storage and processing. Two of the core libraries in Apache Hadoop are Hadoop Distributed File System (HDFS) and Hadoop MapReduce.

By default, Hadoop uses a java-based data processing framework called MapReduce to work with the data itself. MapReduce runs a series of jobs which fetch data from the file system as needed.

The default file system employed by Hadoop is the HDFS, although it can use other file systems as well. HDFS works by splitting files into large chunks and distributing them to worker nodes in a cluster. Each cluster consists of multiple worker nodes and a single master node. The worker nodes house both the chunks of data and the portions of the application software which make use of the data chunk. This combination maximizes the computational speed and efficiency of the HDFS (Hadoop, 2013).

2.9.2.1. HDFS Architecture

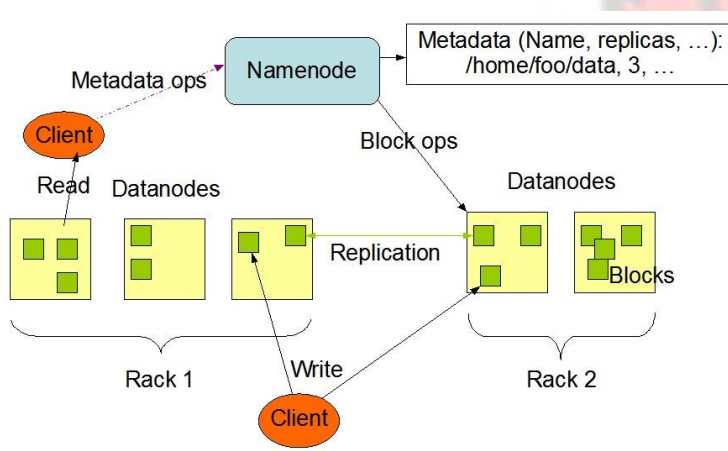


Figure 2.13 – Apache Hadoop Architecture (Hadoop, 2013)

Figure 2.13 shows the Hadoop architecture. A small Hadoop cluster consists of a single master for Job Tracking, Task Tracking, NameNode and DataNode, and multiple worker nodes for Data and Task Tracking. For larger clusters, the nodes in an HDFS are managed by a dedicated NameNode server.

Hadoop operates by splitting files into large chunks and distributing them to the worker nodes in the cluster. It then transfers packaged code to the worker nodes to process the data in parallel. The combination of both data and code on the same node makes computations faster and more efficient than in the alternative supercomputer architecture that relies on a parallel file system (Natarajan, 2012).

2.9.2.2. Erasure Protection

To guard against data loss, Hadoop creates copies of each data chunk and stores the copies in different nodes. This significantly increases the amount of disk space required to store data.

2.9.2.3. Data Encryption

Hadoop permits the use of encryption but recommends plain/raw formats for the files stored in the file system in order for applications to work easily with the file.

2.9.2.4. Data/Resource Location

Hadoop can be deployed in a company's data center or on the cloud. Deploying on the cloud eliminates costs of hardware and specific setup expertise. Vendors such as Microsoft, Amazon, Google, Oracle and IBM currently offer cloud services running Apache Hadoop. In either case, whether using the onsite data center or deploying on the cloud, all the data chunks can be found in one company's servers site (DeZyre, 2016).

2.9.2.5. Data Access

In the case of onsite deployment, the data can be accessed only by the owner of the data. For cloud deployment however, both the data owner and the service provider have access to all the chunks of the data as well as the metadata required to put all the chunks together to reconstruct the full file.

2.9.2.6. Ownership

The ability of the cloud service provider to piece together all the chunks of the data without the express permission of the data owner raises questions as to the actual ownership of the data since the service provider has just as much privileges with the data as does the uploader.

2.9.2.7. Use of the data/resource

With full access comes the ability to use the data for whatever purposes they deem fit. In this regard, the service providers are able to put the data/resource to use for their own benefit without even notifying the uploader.

2.9.3. Backblaze B2

According to BackBlaze (2017), Backblaze is an online data storage service that offers two services; a reliable and affordable object storage service and a computer backup service. The computer backup service allows users of the service to upload all the files on their computers to the cloud. The upload can be scheduled or continuous.

2.9.3.1. Architecture

According to BackBlaze (2015a), all data that is uploaded to Backblaze's servers is first split to 16 shards and 8 parity shards are created using the Reed-Solomon erasure correction algorithm (Figure 2.14). The 24 shards are then stored in separate storage pods, each pod being in a separate cabinet to increase resilience in case of power loss to a cabinet. Backblaze runs only one data centre (Figure 2.15) which is where all data uploaded to the Backblaze cloud is stored.

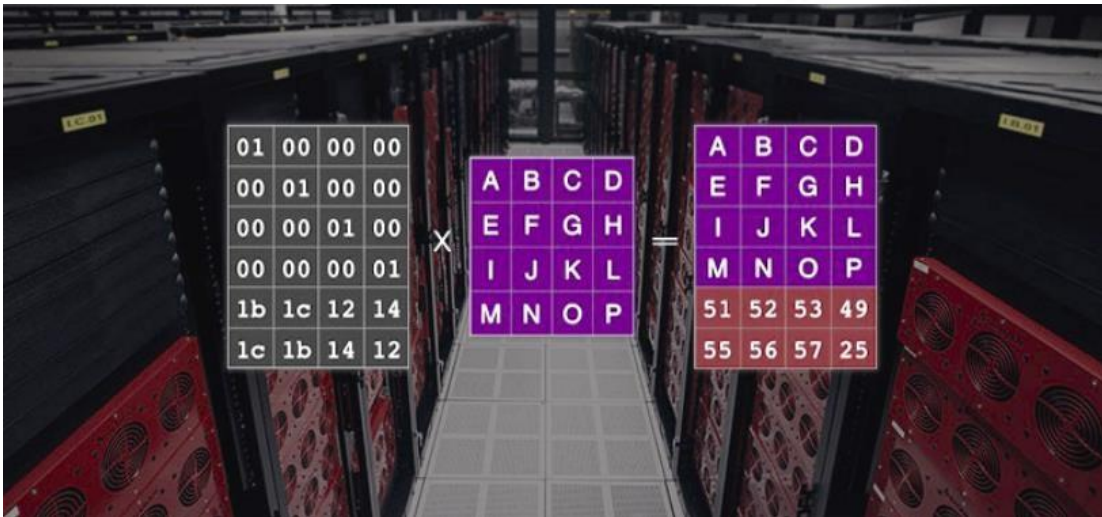


Figure 2.14 – Backblaze B2 System (BackBlaze, 2015a)

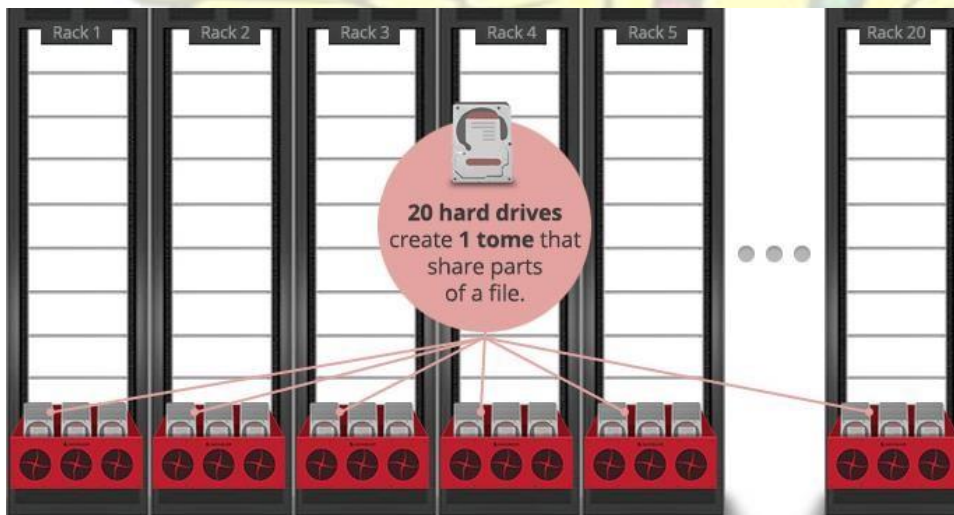


Figure 2.15 – Backblaze B2 System Architecture (BackBlaze, 2015b)

2.9.3.2. Data Encryption

Backblaze uses a combination of AES (Advanced Encryption Standard) and SSL (Secure Socket Layer) to secure client data.

2.9.3.3. Data/Resource Location

All data uploaded to Backblaze is stored in their data center. All the shards are in the same location, accessible by Backblaze.

2.9.3.4. Data Access

The data is fully accessible by the service provider since all the data shards are located in the service provider's data center.

2.9.3.5. Ownership

The full access that the service provider has to the file can create a confusion with regards to the ownership of the file since the service provider can do all the things that the uploader can do with the file.

2.9.3.6. Use of the data/resource

The cloud service provider has all it takes to reconstruct the file and use them to the benefit of the service providers.

2.10. Review of Existing Cloud Storage Security Architecture

2.10.1. Subscriber → Provider (Direct) Model

The prevalent model among users of cloud storage is the direct link to the CSP via the web interface or the desktop/mobile client as shown in figure 2.16. CSPs such as Google Drive, Dropbox and Box all provide an interface through which the client can upload or download files, and in the case of desktop or mobile interfaces, synchronize files and folders on their desktop or mobile device. In the direct model, data security while the file is being transferred and while it is residing in the cloud storage space, is the responsibility of the CSP. Different CSPs implement security in a different ways but the most common approach is to split the file into chunks on the subscriber's end, encrypt the chunks, then transfer the chunks individually to the provider's infrastructure over the internet. The following section reviews how some of the most well-known CSPs implement data security.

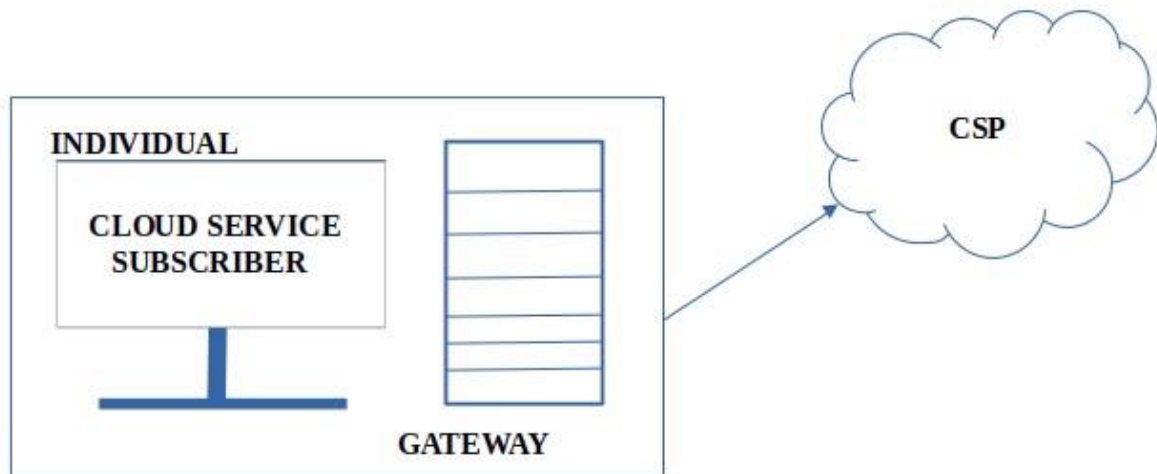


Figure 2.16 - Direct Model of Subscriber-CSP interaction

2.10.1.1. Dropbox Business

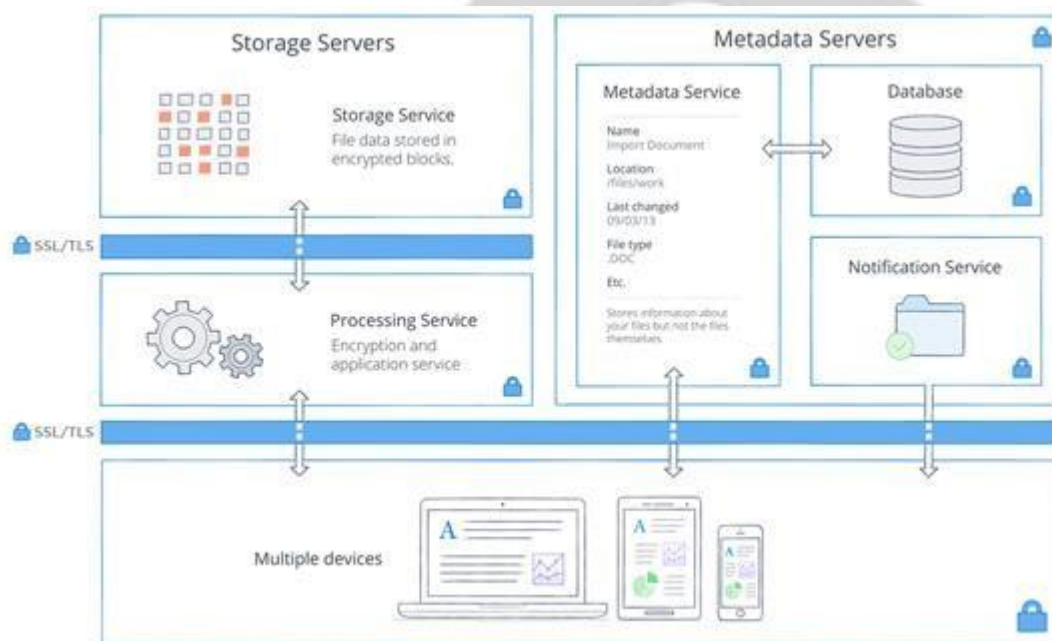


Figure 2.17 - Security Architecture of Dropbox (Dropbox, No Date)

The Dropbox architecture comprises of four services. The four services work together to facilitate the security and ensure the integrity of files sent to Dropbox for safekeeping. The four services are the Processing Service, Storage Service, Metadata Service and the Notification Service (Figure 2.17).

The Processing Service operates between the client side application and the storage service. It works by breaking the file to be synchronized into blocks. The service proceeds to encrypt each block after which the modified blocks are then synced.

The Storage Service operates on the server side. This service works by fetching each file that was uploaded to the server. The files are retrieved based on their hash value and an additional layer of encryption is provided for all the blocks of the file.

The Metadata Service is responsible for storing additional information such as file name and type, in a discrete storage location separate from the file blocks. The metadata provides a record of the files associated with a client's account.

In addition, regular testing and auditing are done on the application and network by teams of internal security specialists as well as third-party specialists to ensure the security of the back-end network.

2.10.1.2. Google Drive

According to TipTopSecurity (2016), the Google Drive client side application first encrypts the data to transfer using Transport Layer Security (TLS) before uploading the data to Google servers. This provides security for the data while it is in transit using the same standard that browsers use to access secure (Hypertext Transmission Protocol Secure – HTTPS) websites.

After the data reaches Google servers, it is decrypted and re-encrypted using 128-bit AES keys. The encryption keys are themselves encrypted with a rotating set of master keys, adding an additional layer of security to the data.

Furthermore, all metadata associated with the uploaded data are encrypted as well.

However, Google operates a privacy policy that allows them to effectively view and use all data uploaded to their servers, as they see fit. In effect, uploading files to Google Drive is comparable to transferring ownership of the file to Google.

2.10.2. Subscriber -> Cloud Access Security Broker -> Cloud Storage Provider (Indirect) Model

In this model, the subscriber adds to data security by involving a Security-As-A-Service (SECaaS) system in the cloud storage setup (Figure 2.18). Cloud Access Security Broker (CASB) systems are SECaaS implementations that function as a software guard for the data that moves around within and

out of an organization. A CASB acts as an independent intermediary between a cloud subscriber and cloud provider. CASB's are on-premises or cloud-hosted software that sits between cloud subscribers and CSP's to enforce security, compliance, and governance policies for cloud usage (SkyHigh, 2017). By employing a CASB introduces an extra layer of security in the cloud environment which gives the subscriber security assurance and install some level of trust (DoubleHorn, 2017). CASB systems enforce the regulations on what data can be transferred in and out of the organization, especially to Cloud Servers (Rubens, 2017).

CASBs support a varying set of functions including, but not limited to, visibility into cloud usage within the organization, enforcement of compliance with the organization's regulations for cloud interaction, protection from external malware and a way to ensure that data is stored in the cloud securely. Notable CASBs are Forcepoint CASB, Skyhigh Networks, Cisco Cloudlock and Microsoft Cloud App Security. By way of securing data sent to the cloud, CASBs often encrypt the organization's data before upload. In cases where the data in the file is marked as too important to risk its contents being disclosed, the CASB protects the organization by preventing the upload of the file to the Cloud Storage Provider. Figure 2.19 is an example implementation of CASB via Cloud Proxy.



Figure 2.18 - Indirect Model of Subscriber-CSP Interaction Using Cloud Access Security Broker

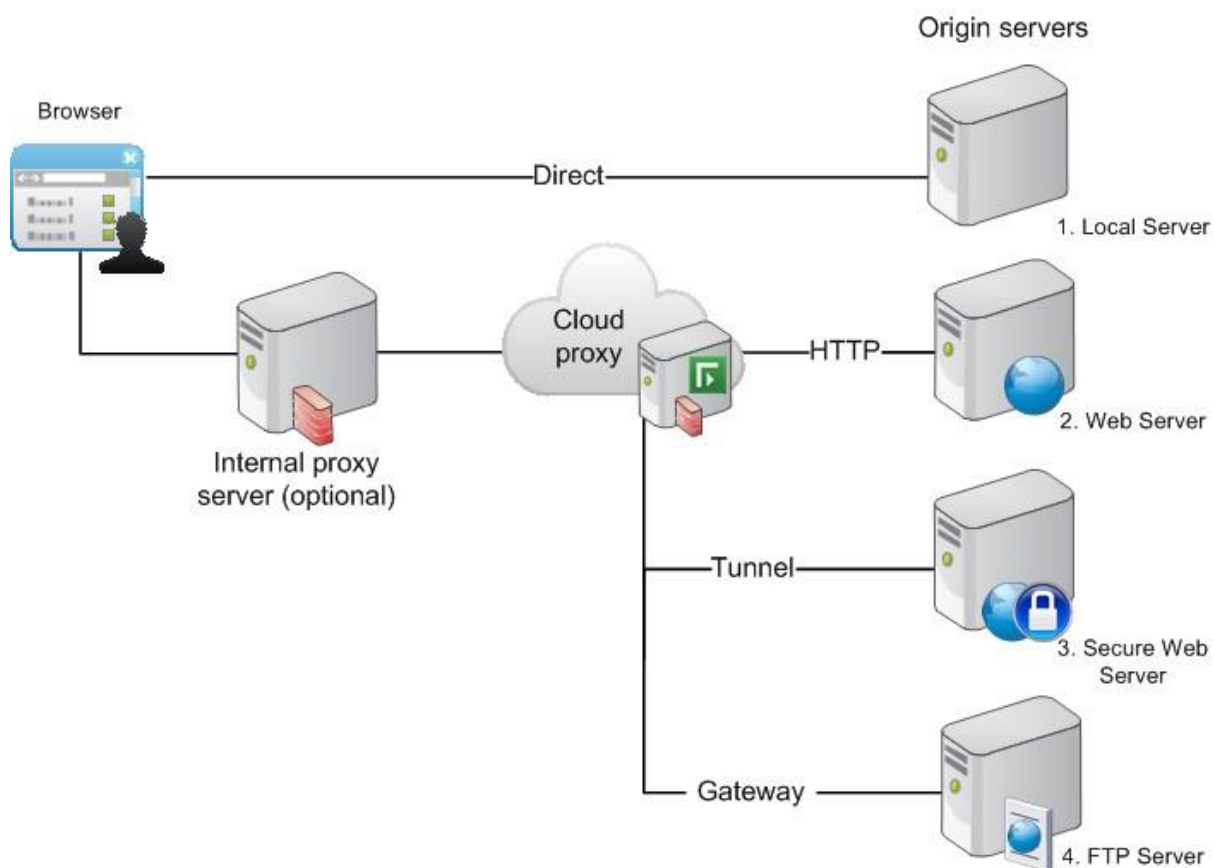


Figure 2.19 - Example Implementation of Cloud Access Security Broker via Cloud Proxy (Forcepont, 2017)

CHAPTER 3

3. METHODOLOGY

3.0. Introduction

In this chapter the methodology for the proposed system for securing files outsourced for cloud storage is presented.

The study employs the design research methodology which enables the development and delivery of new solutions that help to understand human needs and meet them thereby improving livelihoods. Design research enables researchers to understand and incorporate behaviour pattern and define a problem in a context that focuses on achieving a set result (Lee, 2012). This research methodology is adopted as it gives designers and client's clearer understanding of the problem and provides better insight into the problem at hand while providing answers to the most fundamental questions faced throughout the process. It provides answers to questions as:

- What is the correct product or service to design?
- What characteristics should it have?
- Is it the solution working as intended?

Design research doesn't necessarily have to lead to a solution but should generate some good and feasible ideas (Freach, 2011).

The study proposes a framework for securing files outsourced for cloud storage dubbed **Cloud Data Distribution Intermediary (CDDI) that is implemented at the cloud subscriber gateway**. The CDDI framework uses erasure coding, data dispersal, and encryption to secure files. The framework has a client side (CDDI Client) which is implemented on the cloud subscriber's gateway system to encrypt and split the subscriber's data into chunks of data fragments and distribute them randomly to the subscribers selected multiple CSP storage infrastructures. There is also a server side (CDDI Metadata Server) that holds metadata information for all the files that the subscriber uploads to the cloud. The CDDI framework seeks to secure subscribers data from threats such as CSP using subscriber's data, the subscriber not knowing where (which countries) their data is located, the CSP claiming ownership of the subscriber's data, and also not knowing who has unauthorised access to their data. The CDDI is developed into a software system by following a Plan-Driven Incremental software development approach in which system increments are identified and planned well in advance before development. This process method is adopted for the system development because it enables rapid development of the system due to its feature of permitting the software process activities of specification, development and validation to run concurrently. In addition the approach enables the development and delivery of versions of the system prototypes that are introduced to end-users for their rapid feedback (Sommerville, 2011). An Experimental Lab set-up using a very high-spec PC (64-bit Intel Core-i7 CPU running at 3.60GHz, 12.0GB RAM) and Laptop (64-bit Intel Core-i7 CPU running at 2.20GHz, 8.0GB RAM) and requiring a stable computer network infrastructure is employed for the study. The JAVA, SQL, and PHP, software development tools are used to **develop the proposed CDDI framework into a software system dubbed 'SecureMyFiles (SMF)' that is installed on the cloud subscriber gateway system**. The implementation details for the SMF system are presented later in Chapter 4 of this thesis.

3.1. The Proposed Architecture

Figure 3.1 is the overall architecture of the proposed solution for securing data outsourced for cloud storage.

Subscriber → Cloud Data Distribution Intermediary (CDDI) → Cloud Storage Provider

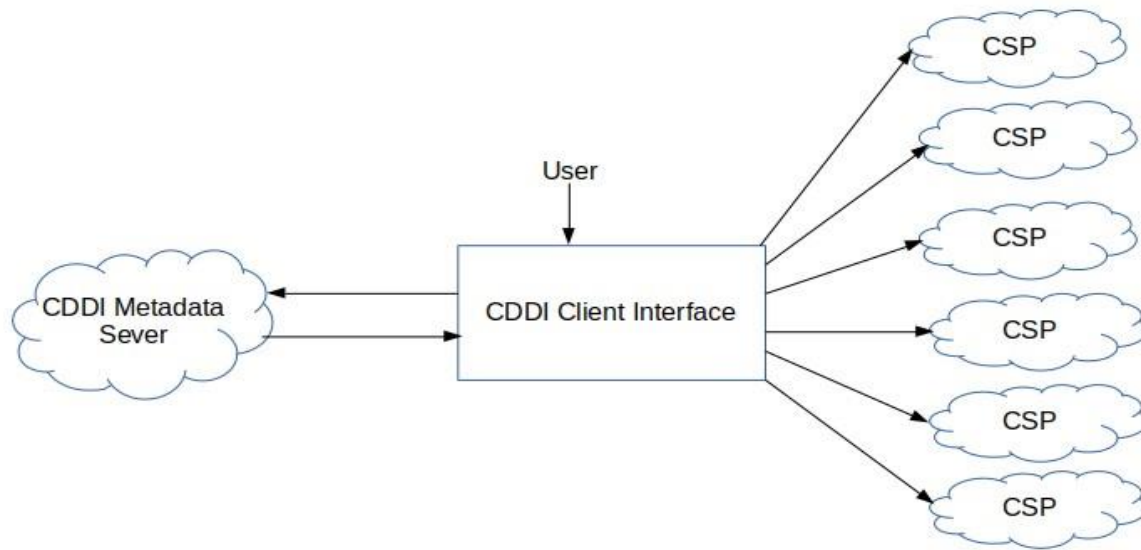


Figure 3.1 - Proposed Model for Cloud Data Storage Using CDDI

3.2. Cloud Data Distribution Intermediary (CDDI)

The study proposes a new indirect model of interaction between the subscriber of a cloud storage service and the Cloud Storage Provider. The study proposes that a software framework intermediary should be introduced into the data transfer transaction to inject a high degree of security into the data storage transaction. The intermediary would be responsible for ensuring that the subscriber's data is protected at various levels from the diverse threats outlined in the problem statement of this study.

Based on the manner in which the intermediary operates, the study refers to the intermediary as a Cloud Data Distribution Intermediary (CDDI). The operation of the CDDI would involve the following processes:

- Receive data from the user for storage in the cloud
- Obfuscate the name of the file to hide its purpose from malicious persons snooping on the network and hackers who may have gained access to the file information in the subscriber's cloud storage account
- Encrypt the subscriber's data to hide its content from unauthorized persons who may obtain it.
- Distribute the encrypted content of the file in unique pieces to a number of Cloud Storage Providers to prevent the problem of one CSP having access to the entire data.
- Save metadata on each file uploaded in order to retrieve the file when required by the cloud subscriber

3.2.1. Components Of The Cloud Data Distribution Intermediary

An effective CDDI framework should comprise of a number of modules, each performing such tasks as would help to secure the file being stored on the cloud.

3.2.1.1. File Name Obfuscation Module

The file name is hashed using a hashing algorithm as the first layer of system security, to obscure the identity of the file being uploaded. This step makes it difficult for people or software that are sniffing on the network from discovering the true purpose of the file while it is in transit. Similarly, any intruder to the subscriber's cloud account would likewise be confounded by the irregular file name.

3.2.1.2. Data Obfuscation Module

As a second layer of security, the contents of the file are transposed using the encryption function of this study's proposed transposition cipher algorithm which is based on the rotations of the Rubik's cube to generate the cipher text.

Initialization - The first activity to perform in using Rubik's Cube as a transposition model is to prepare the six faces of the cube to receive the cleartext (Figure 3.2a and Figure 3.2b).

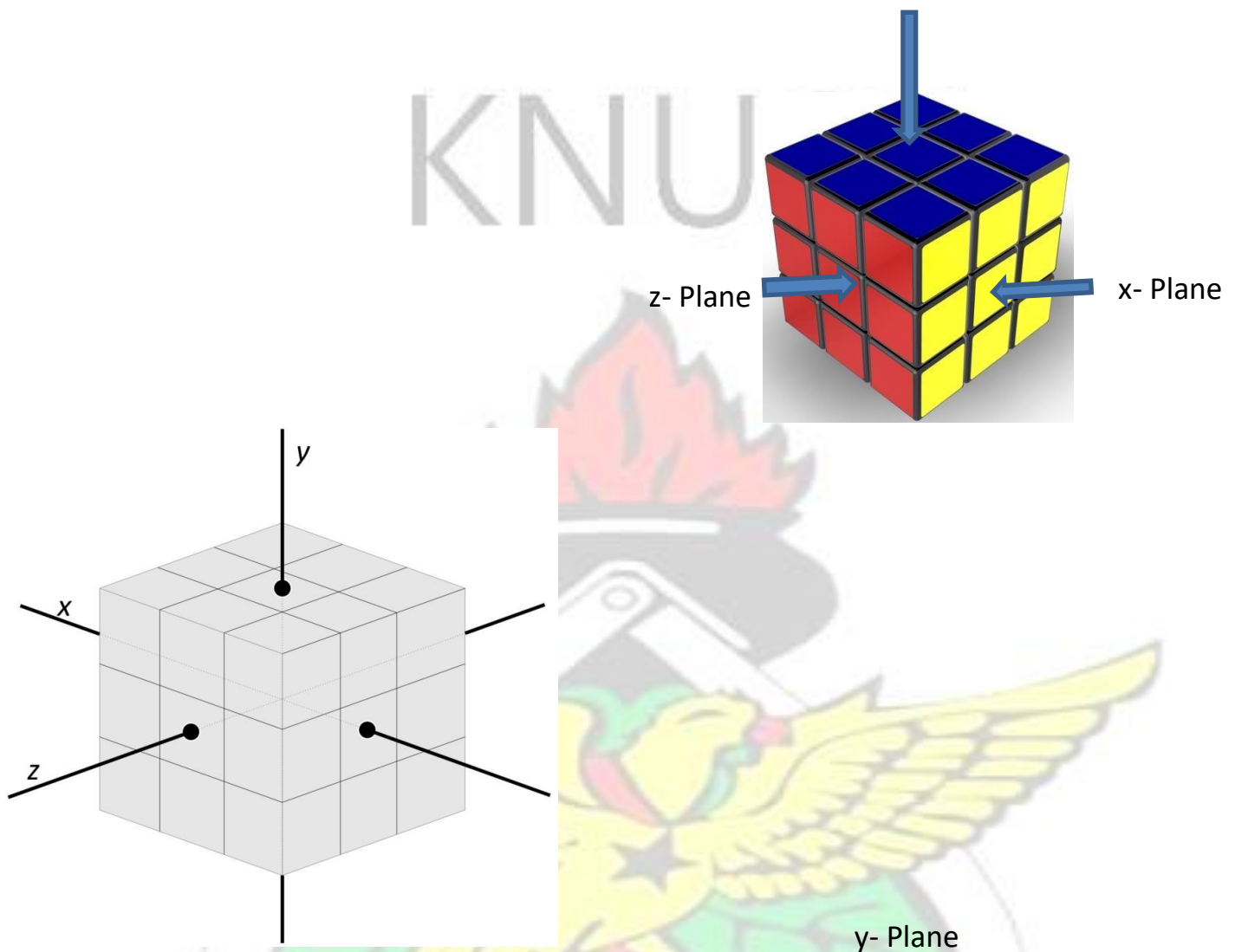


Figure 3.2a - Six faces of the Rubik's Cube

The sizes of the square grid on the faces of the cube are computed so that all the data can be accommodated, with minimum padding. This can be achieved by following the algorithm in Listing 3.1

1. Take the integer ceiling from the division of the length of the data by 6.
2. Take the integer ceiling of the square root of the result from (1).
3. Define 6 two-dimensional arrays to function as the faces of cube using the result from (2) as both dimensions of the array.

Listing 3.1 – Setting up a minimum padding cube **Initialisation**

of the Cube with TEXT

- Assuming the message to be transported (the plaintext) is:

“As a second layer of security, the content of the file”

Algorithm

1. Number of Characters = 54

Take $54 \div 6 = 9$

2. Take $\sqrt{9} = 3$

Face 1 (Front)

A	s	_
a	_	s
e	c	o

Face 2 (right side)

n	d	_
l	a	y
e	r	_

Face 3 (back)

o	f	_
s	e	c
u	r	i

3. Create array with dimensions = $6 \times 3 \times 3$ (i.e.) array [6][3][3]

t	y	,
_	t	h
e	_	c

o	n	t
e	n	t
s	_	o

f	_	t
h	e	_
f	i	l

Face 4 (left side)

Face 5 (Top)

Face 6 (Bottom)

Figure 3.3b - Six faces of the Rubik's Cube initialized with data

After the cube has been created the cleartext are copied onto the faces of the cube sequentially as shown above, from the first face to the sixth. The remaining cells on the face of the cube are filled with the null character or zero.

Generation of Rotation Sequence from the Key - The key for encrypting the data is transformed into a sequence of rotations in order for the encryption to be performed. Two things are needed to perform one rotation – the plane in which to perform the rotation and the index of the row or column on which

the rotation is to be done. Any algorithm which can translate the key into a sequence of rotations is useful at this stage of the transposition process. The algorithm used is as below.

1. Take the Key = “hippopotamus”
2. Take the **SHA1** of the key

$$= \text{a1219e634d04b405d90f13505c4d36578dc97241}$$
3. Take the ASCII value of each character in the **SHA1**
 - a) Determine the plane (x, y, or z) for the rotation = $\text{char} \% 3$
 - b) Determine the index for the rotation $(\text{char}^4 + \text{char}^3 + \text{char}^2) \% \text{cube size (which is 3 for above example)}$
4. Save the result from 3(a) as the plane for the rotation and the result from 3(b) as the index on the plane at which the rotation should be done.

Rotation of the Cube - To mimic the rotation of the cube, strips of data is copied from one face onto another in a set pattern (Figure 3.3a and Figure 3.3b). The pattern in Listing 3.2 is one of several patterns that can be used to implement the rotation activity. This pattern visualizes faces 1, 2, 3, 4, 5 and 6 as the front, right side, back, left side, top and bottom faces respectively. n is the number of row (and columns) of each face of the cube.

To rotate a strip in row i clockwise in the y-axis (**Plane 0**)

1. Copy the data in row i of face 1 to a temporary location.
2. Replace the data in row i of face 1 with the data in row i of face 2.
3. Replace the data in row i of face 2 with the data in row i of face 3.
4. Replace the data in row i of face 3 with the data in row i of face 4.
5. Replace the data in row i of face 4 with the data being held in the temporary location.

To rotate a strip in column i clockwise in the x-axis (**Plane 1**)

1. Copy the data in column i of face 1 to a temporary location.
2. Replace the data in column i of face 1 with the data from column i of face 6.
3. Replace the data in column i of face 6 with the data from column $(n - i)$ of face 3, in reverse order.

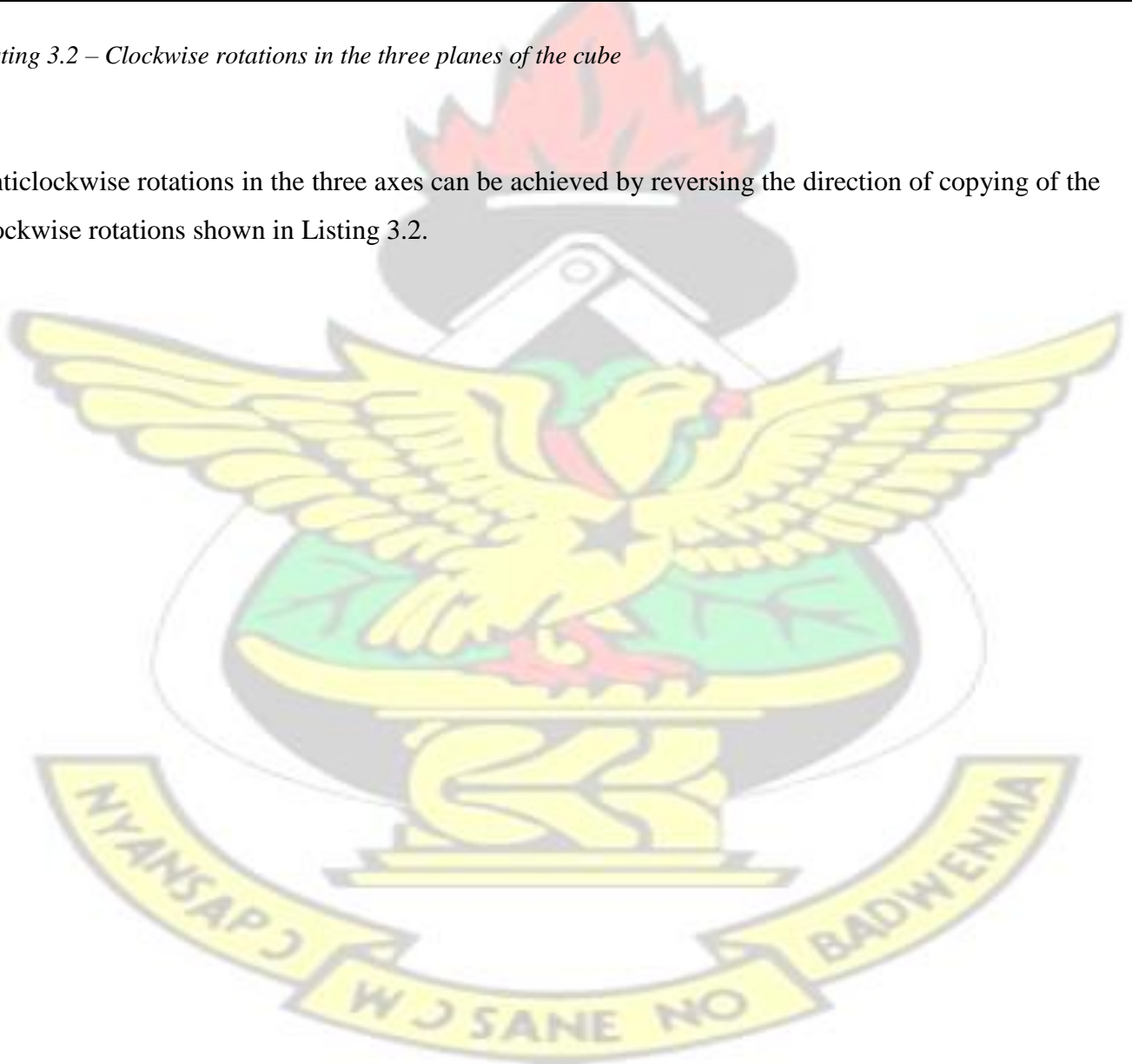
4. Replace the data in column $(n - i)$ of face 3 with the data from column i of face 5, in reverse order.
5. Replace the data in column i of face 5 with the data being held in the temporary location.

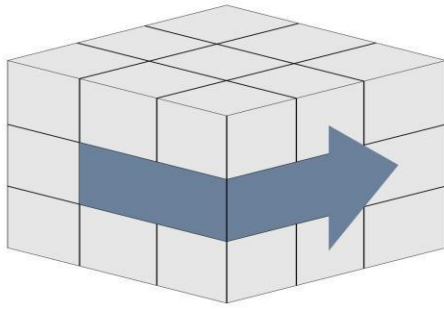
To rotate a strip in column i clockwise in the z -axis (**Plane 2**)

1. Copy the data in column i of face 2 to a temporary location.
2. Replace the data in column i of face 2 with the data from row i of face 6, in reverse order.
3. Replace the data in row i of face 6 with the data from column $(n - i)$ of face 4.
4. Replace the data in column $(n - i)$ of face 4 with the data from row $(n - i)$ of face 5, in reverse order.
5. Replace the data in row $(n - i)$ of face 5 with the data being held in the temporary location.

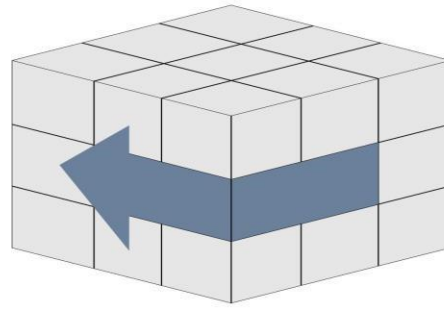
Listing 3.2 – Clockwise rotations in the three planes of the cube

Anticlockwise rotations in the three axes can be achieved by reversing the direction of copying of the clockwise rotations shown in Listing 3.2.

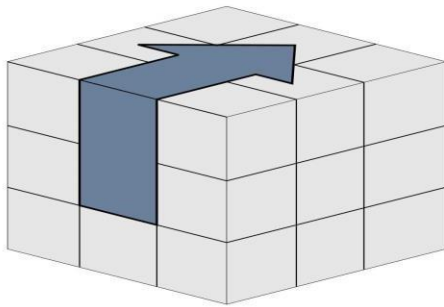




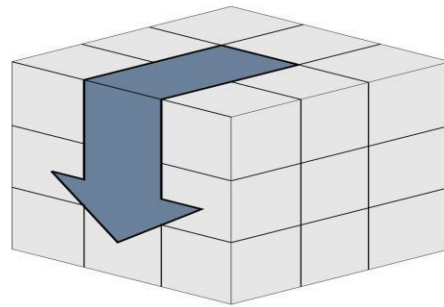
(a) *Clockwise in the y-axis*



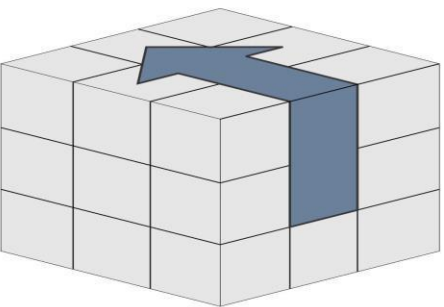
(b) *Anti-clockwise in the y-axis*



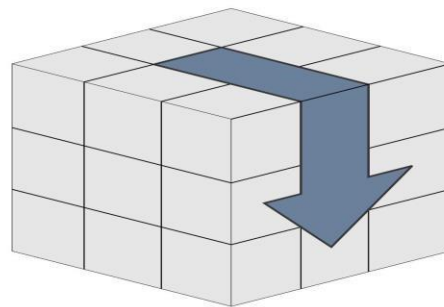
(c) *Clockwise in the x-axis*



(d) *Anti-clockwise in the x-axis*



(e) *Clockwise in the z-axis*



(f) *Anti-clockwise in the z-axis*

Figure 3.4a - Cube rotation pattern



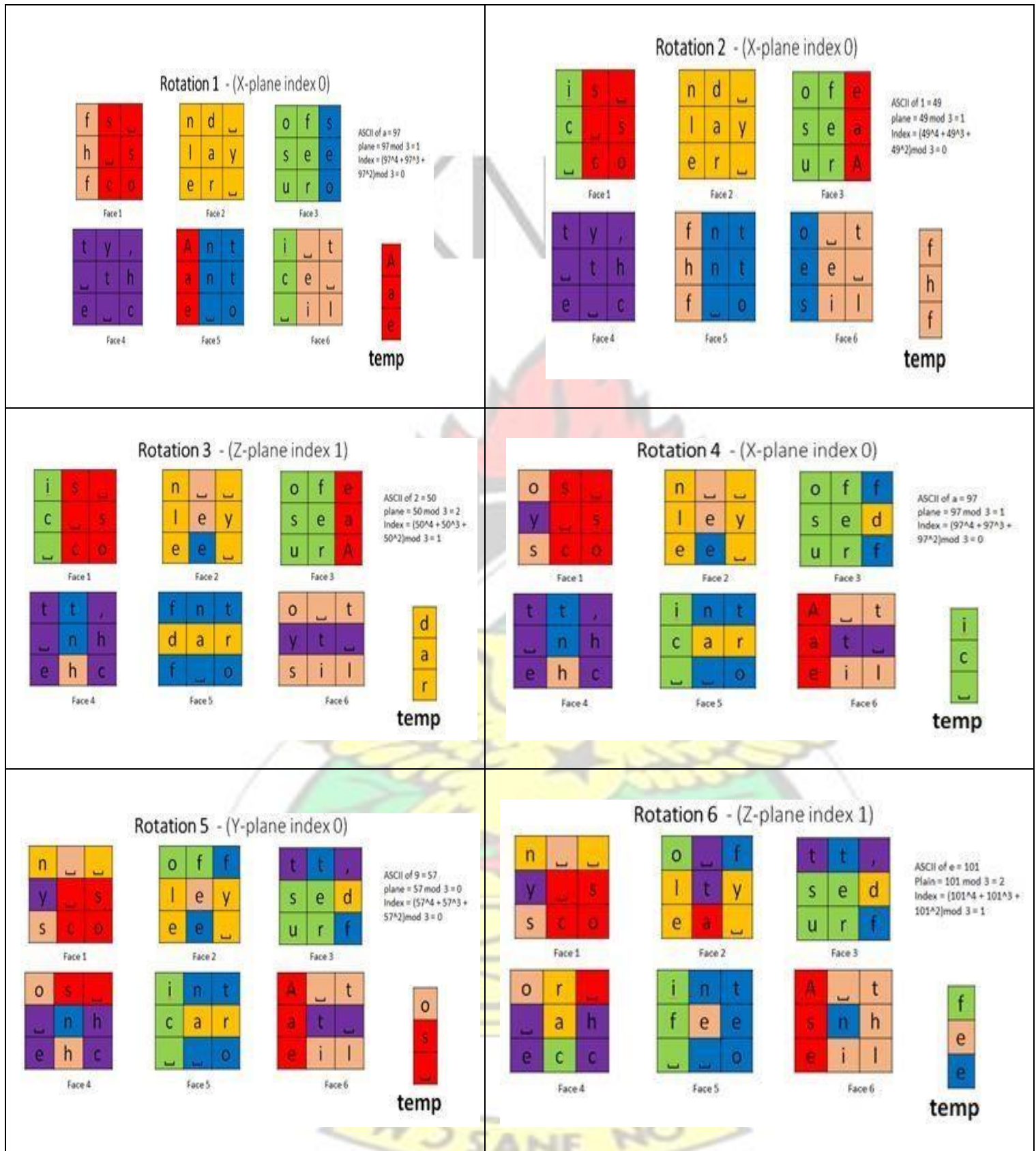


Figure 3.5b - Cube rotation pattern with data

3.2.1.3. Data Distribution Module

The greatest strength of the Cloud Data Distribution Intermediary comes from its use of resilient techniques to distribute the contents of the subscriber's file to multiple Cloud Storage Providers. By so doing, the CDDI is able to mitigate the issue of data ownership on the cloud, as no one CSP has sufficient data to rebuild the file and therefore any claim of data ownership on the part of the CSPs are rendered null by their inability to make any use of the portions of the file in their custody.

The data distribution module comprises of two sub-modules, namely:

1. File Splitting and Erasure Protection Module (FSEPM): This module would be responsible for breaking the encrypted file into a pre-determined number of pieces or shards for subsequent upload to the cloud. To guard against data loss, data corruption as well as Cloud Storage Service down-time, this module makes use of two very resilient techniques to ensure that in most cases, the full file is available to the subscriber when the file is requested. The techniques that the CDDI makes use of, are Reed-Solomon Coding and the newly developed Checksum data recovery technique.
2. Shards Dispersal Module (SDM): This module would be responsible for ensuring that there isn't an observable pattern in how the file shards are sent to the cloud, by scrambling the original order of the shards.

The sections that follow discuss the sub-modules of the Data Distribution Module in more detail.

3.3. File Splitting And Erasure Protection Sub-Module

The proposed system uses one of two Erasure Protection techniques when a file is being uploaded to the cloud. The two techniques are Reed Solomon Coding and Checksum Data Recovery Technique.

3.3.1. Reed Solomon Coding

To enforce Reed Solomon Coding, three polynomials are required, namely:

- i. The irreducible polynomial (also referred to as the generating polynomial)
- ii. The generator polynomial
- iii. The encoding polynomial

As noted in the case of GF(8) earlier in the literature review, for GF(256) or GF(2^8) (the focus of this study), the number of RS codewords generated is obtained as $n = 2^8 - 1 = 255$. Hence for the 32 (8-bits

Symbols) parity shards or (32 Forward Error Correction (FEC) codes) require by this study imply splitting files into 223 (8-bits symbols) data shards. To achieve this, the following are undertaken.

Step-1: A degree 8 irreducible polynomial (a polynomial equivalent of a prime number) in F_2^8 obtained as $P(x) = \alpha^8 + \alpha^4 + \alpha^3 + \alpha^2 + 1 = 285$ [2], (Mathematics Stack Exchange, 2011), is used to generate the GF(256) field elements of (0-255).

Step-2: The generator polynomial which is needed for the generation of the encoding polynomial is defined for the creation of 32 (8-bits Symbols) parity shards or (32 Forward Error Correction (FEC) codes) as follows:

$$G(x) = (x - \alpha)(x - \alpha^2)(x - \alpha^3)(x - \alpha^4)(x - \alpha^5)(x - \alpha^6)(x - \alpha^7)(x - \alpha^8)(x - \alpha^9)(x - \alpha^{10})(x - \alpha^{11})(x - \alpha^{12})(x - \alpha^{13})(x - \alpha^{14})(x - \alpha^{15})(x - \alpha^{16})(x - \alpha^{17})(x - \alpha^{18})(x - \alpha^{19})(x - \alpha^{20})(x - \alpha^{21})(x - \alpha^{22})(x - \alpha^{23})(x - \alpha^{24})(x - \alpha^{25})(x - \alpha^{26})(x - \alpha^{27})(x - \alpha^{28})(x - \alpha^{29})(x - \alpha^{30})(x - \alpha^{31})(x - \alpha^{32})$$

Equation 3.1

Now substituting values of ' α ' in Step-1 with their decimal equivalent from the GF(256) elements (Table A1 – Appendix 1) result with:

$$G(x) = (x + 2)(x + 4)(x + 8)(x + 16)(x + 32)(x + 64)(x + 128)(x + 29)(x + 58)(x + 116)(x + 232)(x + 205)(x + 135)(x + 19)(x + 38)(x + 76)(x + 152)(x + 45)(x + 90)(x + 180)(x + 117)(x + 234)(x + 201)(x + 143)(x + 3)(x + 6)(x + 12)(x + 24)(x + 48)(x + 96)(x + 192)(x + 157)$$

Equation 3.2

Since the addition and subtraction arithmetic operations in GF give the same results, the subtraction operator in [equation 3.1] is replaced with addition in [equation 3.2].

Step-3: The generator polynomial of [equation 3.2] is then expressed in the form

$$G(x) = a_{32}x^{32} + a_{31}x^{31} + a_{30}x^{30} + a_{29}x^{29} + a_{28}x^{28} + a_{27}x^{27} + a_{26}x^{26} + a_{25}x^{25} + a_{24}x^{24} + a_{23}x^{23} + a_{22}x^{22} + a_{21}x^{21} + a_{20}x^{20} + a_{19}x^{19} + a_{18}x^{18} + a_{17}x^{17} + a_{16}x^{16} + a_{15}x^{15} + a_{14}x^{14} + a_{13}x^{13} + a_{12}x^{12} + a_{11}x^{11} + a_{10}x^{10} + a_9x^9 + a_8x^8 + a_7x^7 + a_6x^6 + a_5x^5 + a_4x^4 + a_3x^3 + a_2x^2 + a_1x$$

Equation 3.3

Where the coefficient values

($a_{32}, a_{31}, a_{30}, a_{29}, a_{28}, a_{27}, a_{26}, a_{25}, a_{24}, a_{23}, a_{22}, a_{21}, a_{20}, a_{19}, a_{18}, a_{17}, a_{16}, a_{15}, a_{14},$

$a_{13}, a_{12}, a_{11}, a_{10}, a_9, a_8, a_7, a_6, a_5, a_4, a_3, a_2, a_1$) are used for the generation of the encoding polynomial which is used for the generation of the RS codeword for error detection and recovery in the event of data loss, damage, or alteration in transmission or in storage. The **algorithm** proposed by this study for the generation of the coefficient values is as presented below.

Algorithm for generating the encoding polynomial coefficients - The Generator polynomial is of the form

$$g(x) = (x + \alpha^1)(x + \alpha^2)(x + \alpha^3) \dots (x + \alpha^n)$$

where n is the number of parity data being added.

The coefficients of x in the expansion of $g(x)$ are found using the algorithm below:

for pow = 0 to n

*→ sum all the combinations of the set $\{\alpha^1, \alpha^2, \alpha^3 \dots \alpha^n\}$ that have **pow** elements*

For example, to find the coefficients for $n = 4$, assuming the Galois Field elements are

$$\alpha^1 = a$$

$$\alpha^2 = b$$

$$\alpha^3 = c$$

$$\alpha^4 = d$$

Then the looping process performs the following action

$$pow = 0: \{ \}$$

$$pow = 1: a + b + c + d$$

$$pow = 2: ab + ac + ad + bc + bd + cd$$

$$pow = 3: abc + abd + acd + bcd$$

Thus

Loop $n + 1$ times ($i \rightarrow 0$ to n)

Generate all subsets of the alpha exponents set, of size $(n - i)$

Multiply the components of each of the subsets

Add the products of the subset components

Save the sum of the subsets in a coefficients array

Listing 3.3 - Algorithm to generate the coefficients of the encoding polynomial

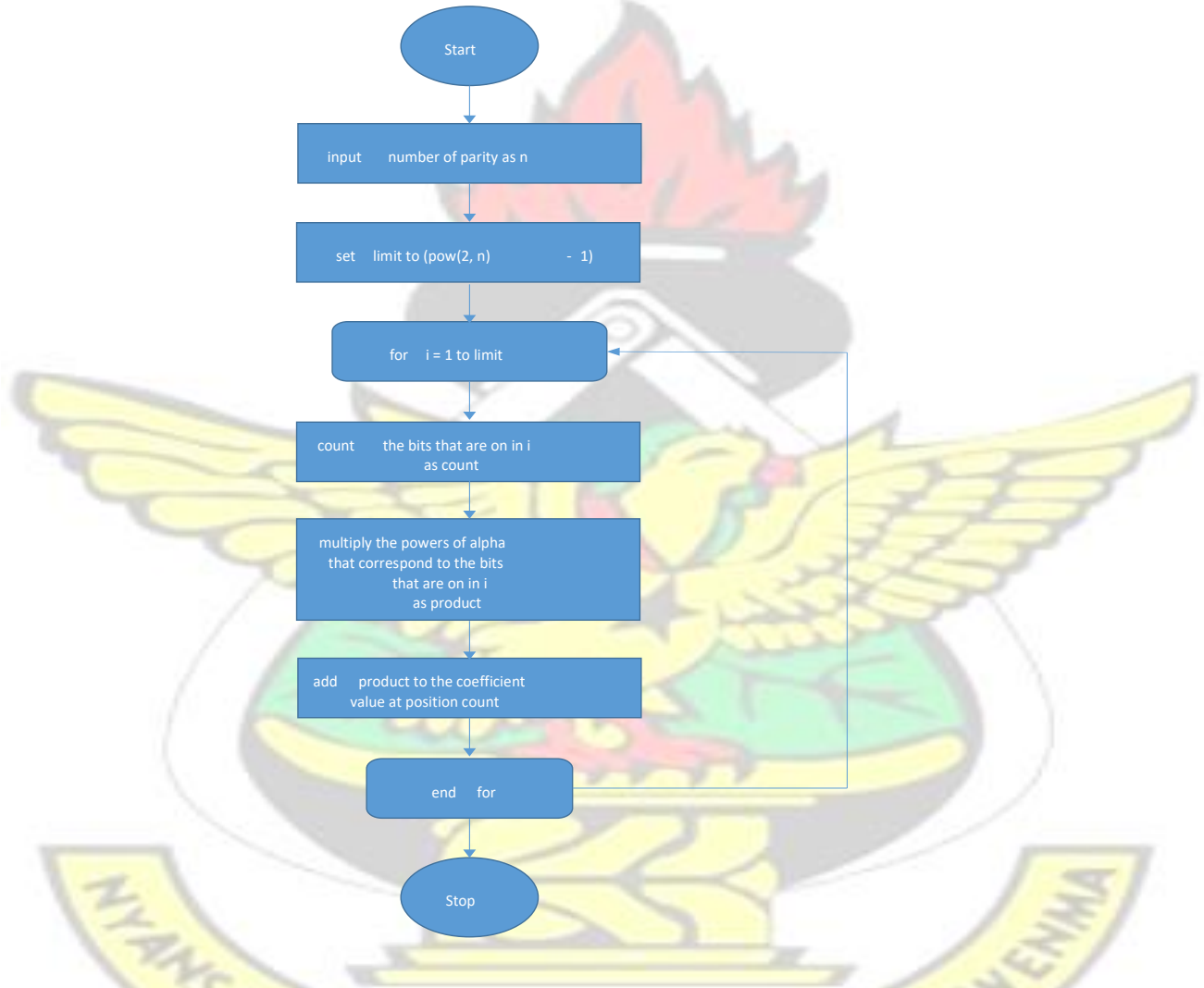


Figure 3.6 - Flow diagram for the algorithm

3.3.1.1. Polynomial arithmetic

As noted in literature under finite field arithmetic, addition and subtraction amount to the same thing in $GF(2)$. Hence, the procedure for adding one polynomial to another involves simply performing an *XOR* operation on the array elements that have the same index. Therefore the algorithm for adding two polynomials is as follows:


```

:: function polynomial_addition(a[ ], b[ ])
::
::   longer [ ] = longer of the two arrays(a, b)
::
::   shorter[ ] = shorter of the two arrays(a, b)
::
::   for i = 0 to length of shorter
::
::       longer[i] = shorter[i] XOR longer[i]
::
::   end for
::
::   return longer
:: end function

```

Listing 3.4 - Pseudocode for polynomial addition

Multiplication is based on the simple principle that

$$ax^n \times bx^m = abx^{n+m}$$

The pseudocode below describes a function to multiply two polynomials:

```

:: function polynomial_multiplication(a[ ], b[ ])
::
::   for i = 0 to length of a
::
::       for j = 0 to length of b
::
::           product[i + j] = product[i + j] XOR galois_multiply(a[i], b[j])
::
::       end for
::
::   end for
::
:: end function

```

Listing 3.5 - Pseudocode for polynomial multiplication

Division is based on polynomial long division. The pseudocode below describes a function to perform polynomial long division.

```

:: function polynomial_division(a[ ], b[ ])

```

```

::   for i = 0 to (length of a – length of b)

::       quotient[i] = galois_divide(b[0], a[i])

::       for j = 0 to length of b

::           a[i + j] = a[i + j] XOR galois_multiply(quotient[i], b[j])

::       end for

::   end for

::   remainder[ ] = subarray of a beginning at index (length of a – length of b)

::   return quotient, remainder

:: end function

```

Listing 3.6 - Pseudocode for polynomial division

The Reed Solomon encoding process requires that the polynomial be shifted a number of degrees up. The pseudocode below describes a function to shift the polynomial a number of degrees up.

```

::function polynomial_shift(a[ ], n)
::   for i=0 to n-1
::       result[i]=0
::   end for
::   for i=n to (length of a+n-1)
::       result[i]=a[length of a-i]
::   end for
::end function

```

Listing 3.7 - Pseudocode for shifting a polynomial by a number of degrees

Reed Solomon Encoding Process - The underlying principle of Reed Solomon Encoding is to adjust the original data so it becomes a perfect multiple of another predefined polynomial called the encoding polynomial. The process is summed in an Algorithm as follows:

- i. Generate an encoding polynomial

- ii. Shift original data polynomial to make space for the parity data (**NB. The shift depends on the number of error correction bits required**)
- iii. Take the remainder from dividing the modified data polynomial by the encoding polynomial. iv. Subtract the remainder from the modified data to generate a perfect multiple of the encoding polynomial.

The resulting modified data polynomial, called the **Reed Solomon Codeword**, can be **tested** later for corruption by checking if it leaves a remainder upon being divided by the encoding polynomial. If there is a remainder, the data is corrupt. An implementation of the algorithm is presented later in Chapter 4.

3.3.1.2. Reed Solomon Decoding

The study uses the Reed-Solomon Decoding process as described in Section 2.8.7. The Syndrome Polynomial (Trench, 2003) which quantifies the error location and magnitude is computed using the equation:

$$S_i = R(a^i) = T(a^i) + E(a^i) = E(a^i) = Y_1 a^{ie_1} + Y_2 a^{ie_2} + \dots + Y_v a^{ie_v}$$

$$= Y_1 X_1^i + Y_2 X_2^i + \dots + Y_v X_v^i$$

Where e_1, e_2, \dots, e_v represents where the error(s) is/are located and the Y_1, Y_2, \dots, Y_v corresponds to the error magnitude.

The **key equation** is of the form $\Lambda(x)S(x) = Q(x)x^n + \Omega(x)$

Where $\Lambda(x)$ represents the locator polynomial, $S(x)$ is the syndrome polynomial, $Q(x)$ is the quotient, x^n is the field polynomial, and $\Omega(x)$ the magnitude polynomial

This study adopts the **Euclid-Sugiyama Algorithm (or Extended Euclidean Algorithm)** for obtaining the locator and error magnitude polynomials needed for solving the key equation. The choice of the Euclid-Sugiyama Algorithm is as a result of its easiness of implementation in software.

3.3.2. Checksum Data Recovery

The study developed a data recovery technique based on checksums, making use of the unique property of the bitwise XOR operator. The technique uses a thorough computation of checksums on sections of the file that is being uploaded. The checksum data can then be used later to recover deleted portions of the file as well as detect and correct errors in the file.

3.3.2.1. Overall architecture of the proposed Checksum Data Recovery Technique

The overall architecture of the proposed **Checksum Data Recovery Technique** is as shown by Figure 3.5. The system has 3 main modules as Data, Compute Parities, and LocateError.

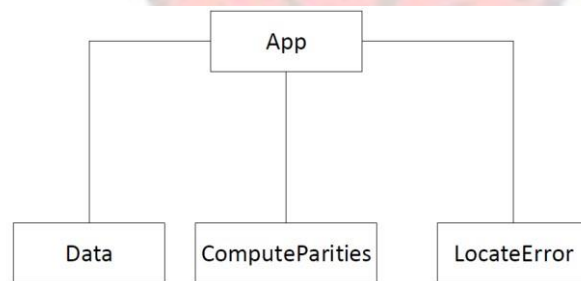


Figure 3.7 - Overall architecture of the proposed Checksum Data Recovery program

The proposed system for protecting data outsourced for cloud storage divides data into several modules (data shards) [Figure 3.6], and parity information (checksum) for each module is computed and stored (Figure 3.7). Section 3.3.2.3 gives further details of how this is achieved. The implementation is presented in chapter 4.

Data corruption is detected and corrected by re-computing the checksum values and comparing with the previously stored value.

3.3.2.2. Modular Representation Of Data

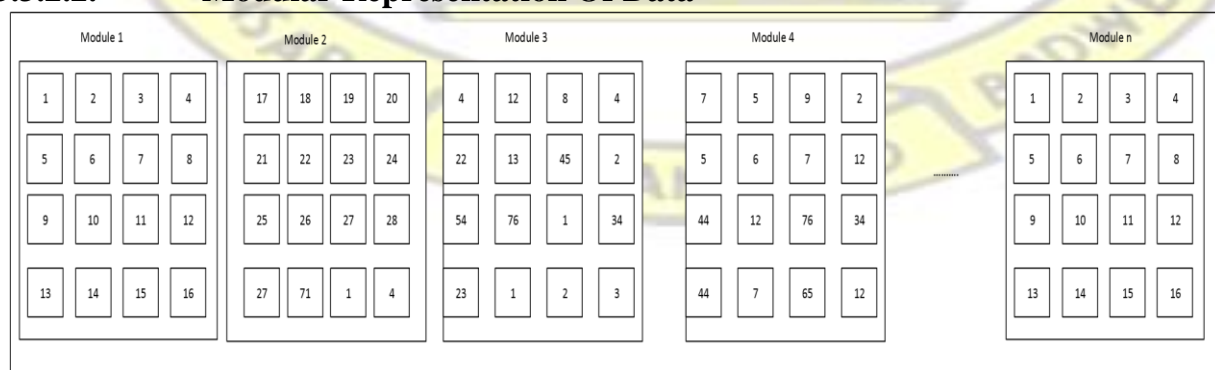
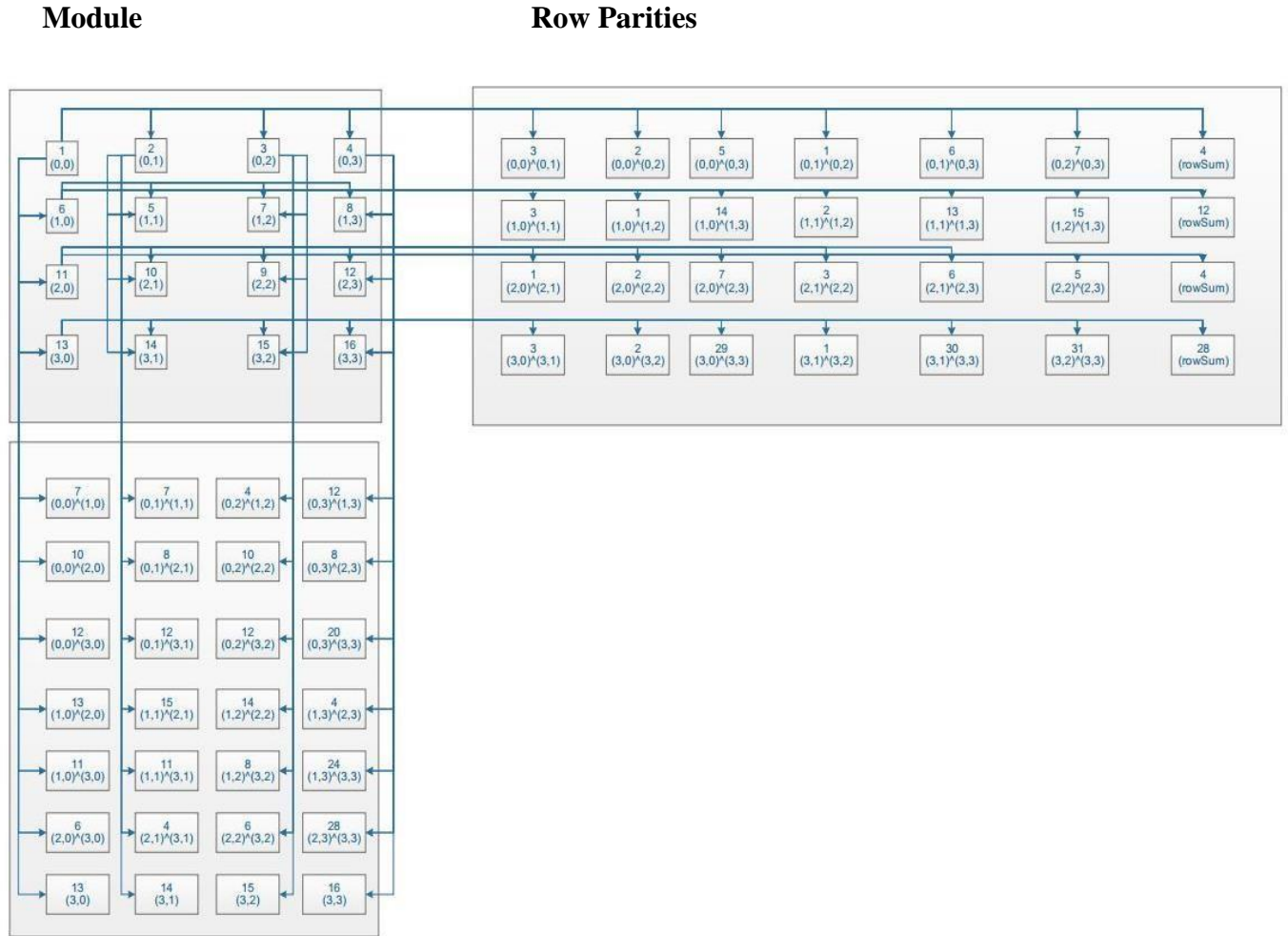


Figure 3.8 - Modular representation of data



Column Parities

Figure 3.9 - Module diagram of the proposed Checksum Data Recovery Program

3.3.2.3. Definition of Terms

Data and Module - A given file of size 'X' (which can be in KB, MB, GB, TB etc.) is computed into a 3-dimensional (3D) array and each entry of the 3D array is a 2-dimensional (2D) array of size 4x4 matrix called a module or data shard (Figure 3.7).

Thus, a module is a 4x4 matrix that has a byte of the data in each entry. This implies a module has a size of 16 bytes.

Therefore to obtain a module, the file of size 'X' is first converted into bytes of data (say 'Y' bytes) and then divided by 16. As the file size may not exactly be a perfect multiple of 16 byte, this may result with Y/16 modules remaining Y mod 16 bytes of the data. The remainder (i.e. the Y mod 16) is padded with zeros to make up a module (16 bytes).

Each module within the data has its own metadata (Row and Column Parity information) which is independent of other modules (Figure 3.7). Therefore a corrupted module does not depend on other modules for recovery.

Thus, the metadata for each module are independent, implying that a corrupted metadata does not affect other module metadata.

The above data representation depicted by Figure 3.6 shows that data can be grouped into modules up to 'n'.

Row Parities are the parities of each row of a module. It is computed by performing the XOR of a value in a row with the values succeeding it in the same row as depicted by figure 3.7.

Column Parities are the parities of each column in a module. It is computed by performing the XOR of a value with the values succeeding it in the same column as depicted by figure 3.7.

3.3.2.4. Architecture of the Data Module

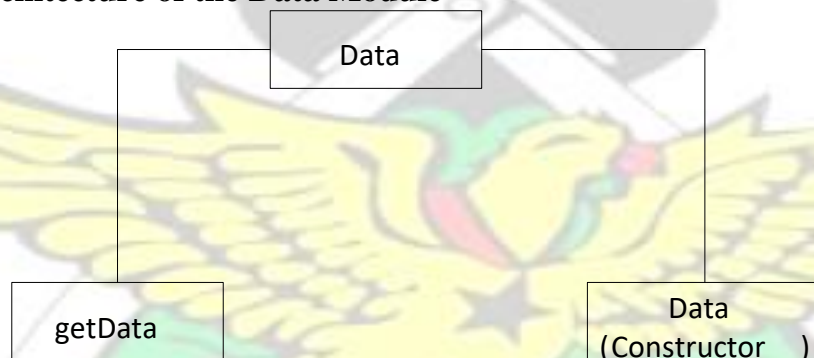


Figure 3.10 - Architecture of the Data Module

The Data module has only two methods: Its **constructor** method and the **getData** method.

The Constructor method converts a file passed into it as an argument into a three-dimensional array and stores it in the array. The **getData** method returns the data to the ComputeParities Module as a three dimensional byte array.

3.3.2.5. Architecture of the ComputeParities Module

The ComputeParities Module has two sub modules: **Computation** and **get**. The sub modules each have three methods.

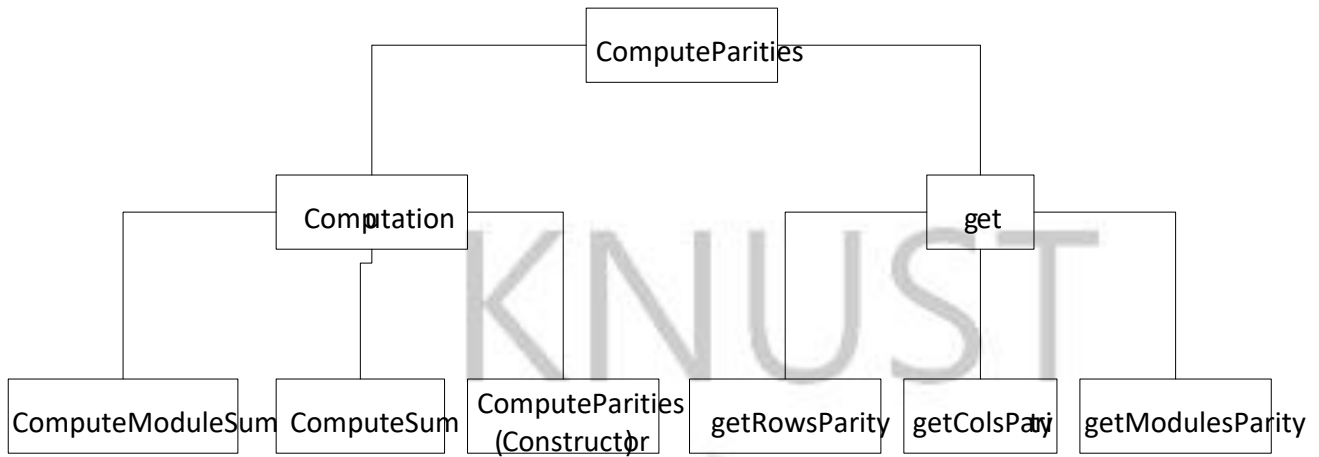


Figure 3.11 - Architecture of the ComputeParities Module

3.3.2.6. Computation Sub Module

The **Computation** sub module which is a sub-module of ComputeParities as illustrated by figure 3.9 has the following three methods:

ComputeModuleSum: This method computes the sum of a module (XOR of all elements in a module as shown by Figure 3.6) and stores it in an array.

ComputeSum: This method computes the parities of the rows and columns (Figure 3.6) of the data that is passed to the Module constructor method.

ComputeParities(Constructor): This method initializes the Module variables that are declared.

3.3.2.7. get Sub Module

The **get** sub module has the following three methods:

getRowsParity: This method obtains the computed Rows parity values of a module (Figure 3.7).

getColsParity: This method obtains the computed Columns parity values of a module (Figure 3.7). **getModulesParity:** This method obtains the computed Modules parity values (Figure 3.7).

3.3.2.8. Architecture of the LocateError Module

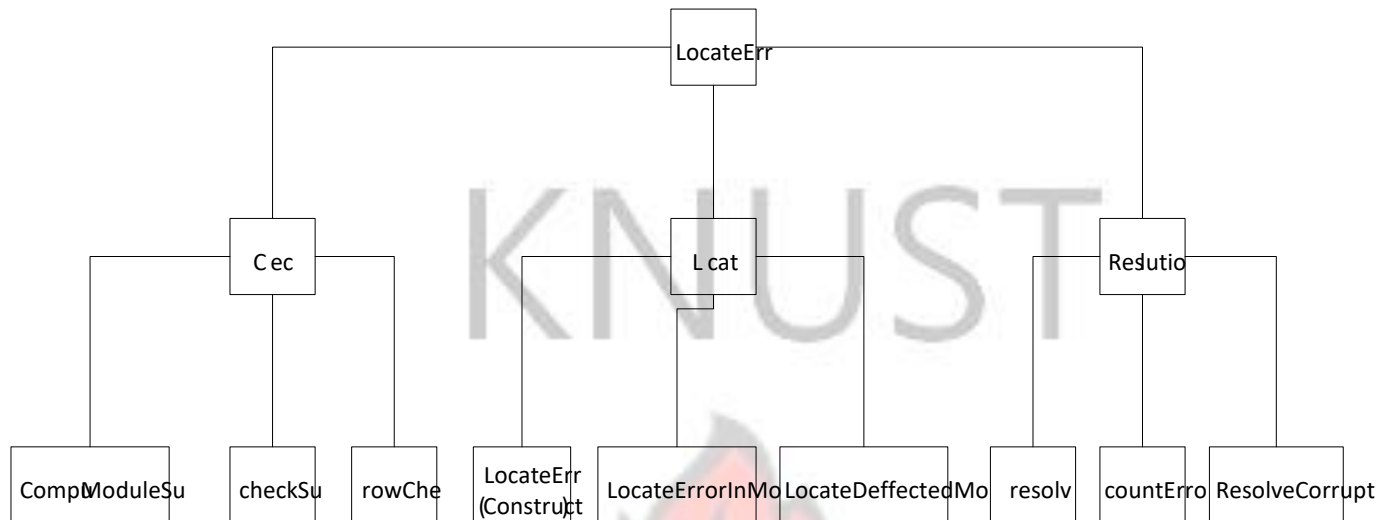


Figure 3.12 - Architecture of the LocateError Module

The LocateError Module has three sub modules, **Check**, **Locate** and **Resolution**. The sub modules each have three methods as shown by figure 3.10.

Check Sub Module: This sub module checks the *integrity* of the data. It has three methods as follows:

ComputeModuleSum: This method computes the sum of the module.

checkSum: This method checks the computed Module sum against a value that is passed to the method.

rowCheck: This method checks the row been worked on for the starting and ending values needed for the resolution.

Locate Sub Module: This sub module has three methods that are used to check for the errors in the module.

LocateError: This is the constructor for the module. It initializes the variables of the module with the arguments that are passed to it.

LocateDeffectedModule: This method checks to see where the error(s) is/are in the data.

LocateErrorInModule: This method checks to see what the errored data in the module is and corrects it.

Resolution Sub Module: The sub module has three methods that are used to restore the data to its original form. **countErrors:** This method is used to check the number of errors that are in a module.

resolve: This method corrects the errors for the module.

ResolveCorruption: This method corrects the errors for the last columns of each module.

3.3.2.9. Flow diagram of the system

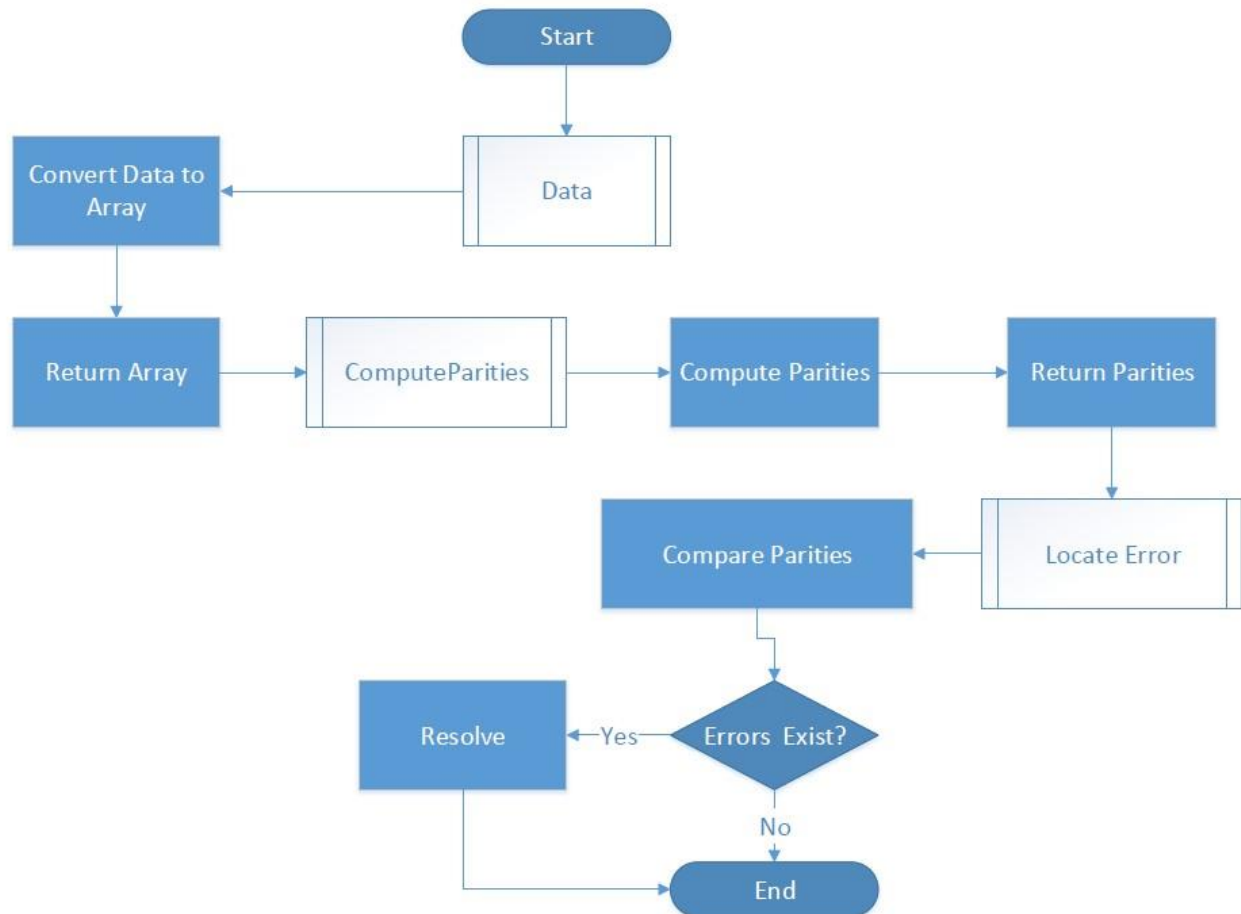


Figure 3.13 - Flow diagram for the Checksum Data Recovery program

3.3.2.10. Activity diagram for the system

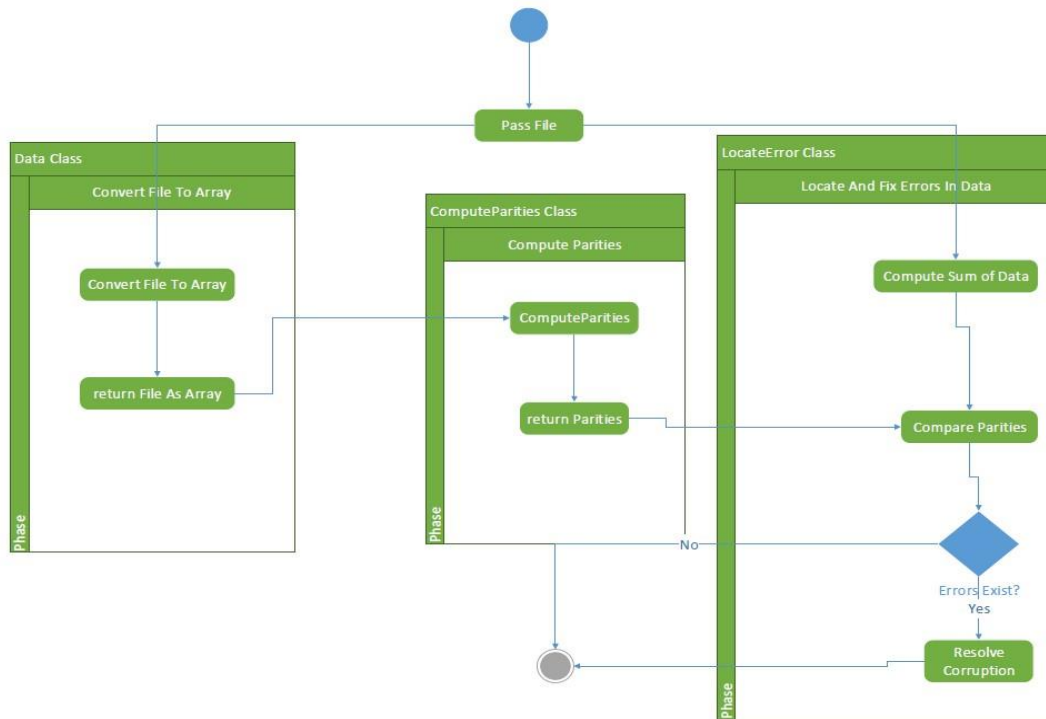


Figure 3.14 - Activity diagram of the Checksum Data Recovery program

3.3.2.11. Sequence diagram for the system

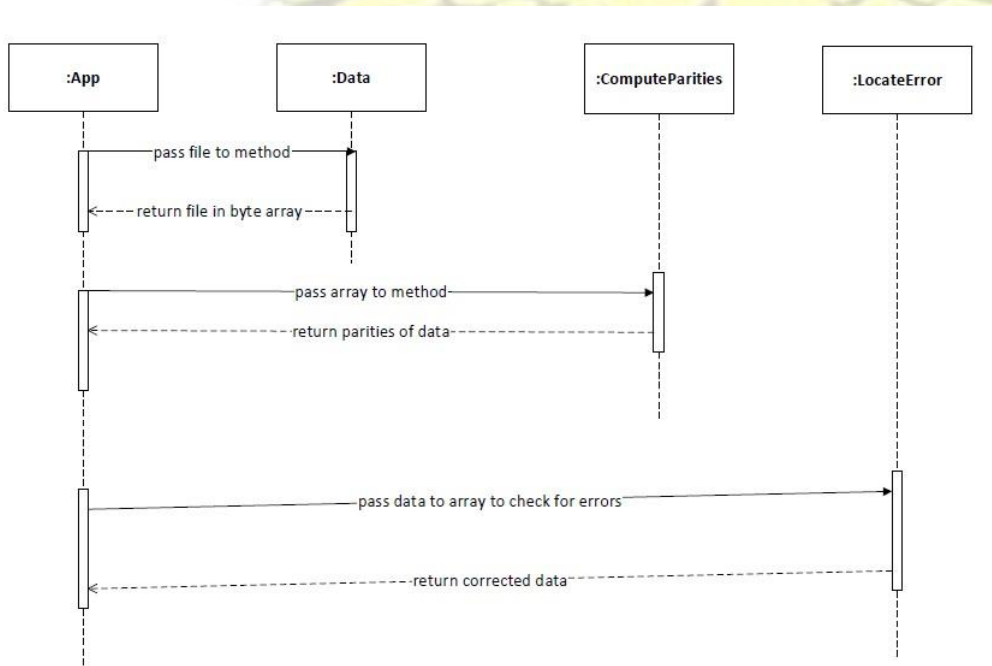


Figure 3.15 - Sequence diagram of the Checksum Data Recovery program

3.3.2.12. Class diagram for the system

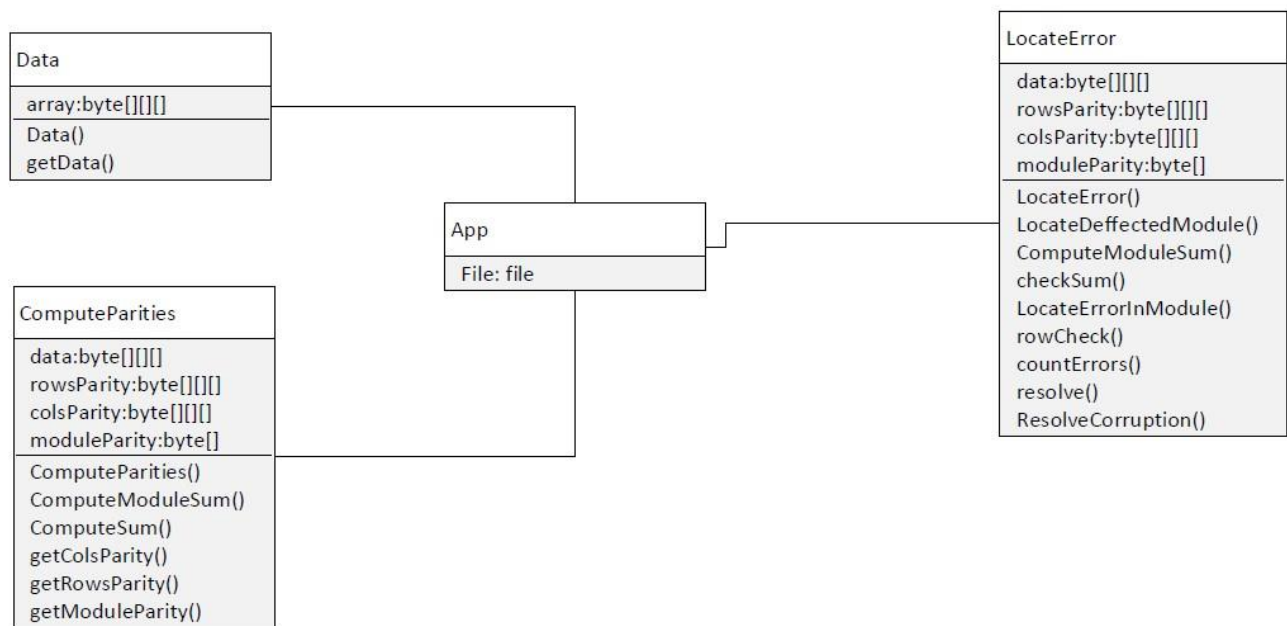


Figure 3.16 - Class diagram for the Checksum Data Recovery program

3.3.3. Shard dispersal module

To improve security and avoid prediction of the destination of the shards, the derived shards after splitting are scrambled before they are forwarded to the various cloud storages. In other words, the transmission of the shards is obscured so that the shards will not be uploaded orderly but rather shuffled among the users chosen cloud storage infrastructure. This is achieved by following the underlying algorithm.

3.3.3.1. Algorithm

- Create an array list containing integer numbers that correspond to the number of the derived shards after file splitting.
- Shuffle the integer numbers in the new list.
- Append these numbers from the shuffled list to the name of the file and use the new name to get the individual shards to be ready for upload.

Implementation of the algorithm is presented in Chapter 4.

3.4. Metadata Module

The proposed system makes use of metadata in the Cloud Data Distribution Intermediary to save data concerning each file uploaded by a user. Two different types of metadata are used. One keeps record of the App user's uploaded files (*user metadata*), and the other keeps track of the uploaded shards to the multiple cloud providers (*file metadata*).

The user metadata has the names of all files that a user has uploaded using the application. It also has the user's hash value which is used for encrypting each file the user intends to encrypt and upload. With the user metadata, the list of files that a user has uploaded can be retrieved and rendered in a view to the user in the application. It thus, relieves the user the burden of keeping track of uploaded files.

The other metadata, dubbed 'file metadata', contains data relating to each of the files. It stores details for each data shard belonging to a file. From the file metadata, a shard's position in the sequence of the shard chunks can be determined. The destination cloud account is also saved in the file metadata. Again, it has other details such as date of upload, and the number of columns which is essential for the Reed-Solomon algorithm. Implementation of the metadata is presented in Chapter 4.



CHAPTER 4

IMPLEMENTATION

4.0. Introduction

The proposed Cloud Data Distribution Intermediary (CDDI) framework is implemented into software using the Java, SQL and PHP programming languages. The software is named SecureMyFiles (SMF) System.

4.1. SecureMyFiles System

The SecureMyFiles (SMF) system has three components as shown by figure 4.1.

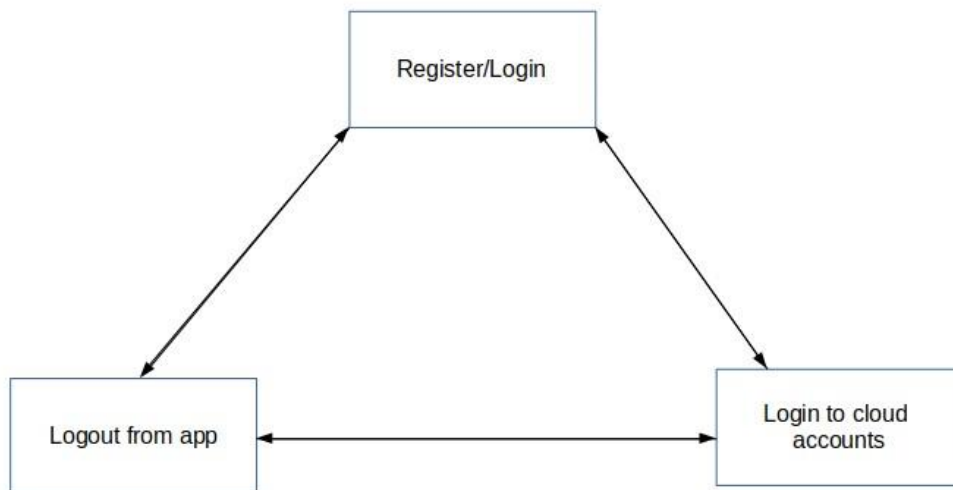


Figure 4.1 - Components of SMF System

4.2. The Login Module

The login module, a sub-component of the SMF System, is composed of the registration stage for a new user of the system, a login stage for an already-registered system user, and a password request stage. The user makes use of the login module to either log into the system or sign up to use the SecureMyFiles system. The interfaces of the login module components are shown in figure 2 and figure 3 in Appendix 2.

4.3. File Upload Module

The file upload module depicted by Figure 4.2 is one of the main sub-components of the proposed system, the SMF System. The user first selects the file to upload regardless of the format or type—e.g. video, audio, text, etc.; whether zipped or unzipped.

The file name is **hashed** using SHA-1 as the first layer of system security to obscure the identity of the file being uploaded. Refer to Section 3.3.1.1 of the methodology and section 4.4 for the implementation in code for the hashing module.

As a second layer of security, the contents of the file are **transposed** using the **encryption function** of this study's proposed **transposition cipher** algorithm based on Rubik's cube (explained in detail in Chapter 3 section 3.2.1.2) to generate a cipher text (an encrypted output). Code implementation of the algorithm is presented in Section 4.5.

As a third layer of security, the encrypted output file is **split** into shards using the **Reed-Solomon coding** technique or this study's proposed new **Checksum Data Recovery** technique presented in Section 3.3 of Chapter 3. The detailed implementation in code is presented at Section 4.8 and Section 4.9.

As a fourth layer of security, the shards upload to a particular cloud account is not done in the order which the splits are produced. Instead, the splits are **distributed** in a **non-deterministic** manner to the user's subscribed selected multi-cloud providers' storage facilities using a shuffling method. Refer to Section 4.10 for the implementation process and code.

A file's **metadata** file is used to keep record of which split chunks are sent to which cloud provider's facility for storage. Finally, the SMF System **user metadata** is updated to keep track of user's uploaded files and to help in retrieval of uploaded files from the various cloud accounts. Section 4.11 presents the implementation of the metadata in code and an explanation of how it works.

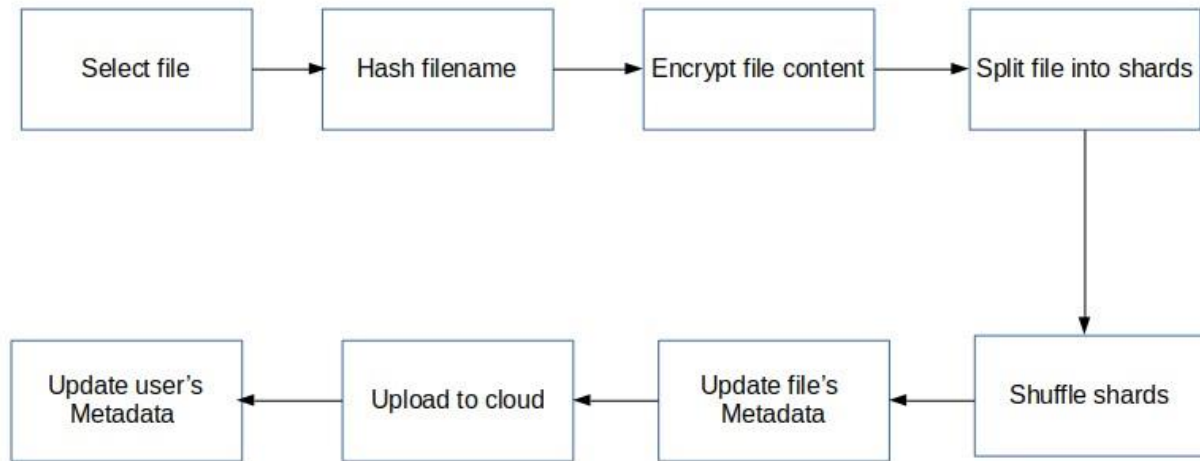


Figure 4.2 - File Upload Module

4.4. Implementation Of The Hashing Method

The system uses the hashing methods in the `java.security.MessageDigest` package to hash the file name. The input to the system's hashing method is a string (the file name). The method returns the digest of the file name as a string. Listing 4.1 is a code snippet of the implementation of the hashing method.

```

public static String getHash(String txt){
    java.security..MessageDigest md =
        MessageDigest.getInstance("SHA1");
    byte[] array = md.digest(txt.getBytes());
    StringBuilder sb = new StringBuilder();
    for (byte anArray : array)
        sb.append(Integer.toHexString((anArray & 0xFF) |
0x100).substring(1, 3));    return sb.toString();
}
  
```

Listing 4.1 - `getHash` method to generate the hash of a string

4.5. Implementation Of The Proposed Transposition Cipher Algorithm Based On Rubik's Cube Transformation

An encryption module that mimics the motions of the Rubik's Cube is designed (Refer to Chapter 3 Section 3.2.1.2) and used to obfuscate the data within the file to generate an encrypted output file (the `cipherText`) as follows:

4.5.1. Initialization

The cube is initialized using the size of the data to be transposed. In order to achieve minimum padding, some computations need to be performed to minimize the size of the cube while still being able to accommodate all the data. The algorithm in Listing 4.2 shows one way of achieving the minimum-padding cube.

```
function initializeCube (data[ ])
dataLength=data→length
initSquare=ceil(dataLength/6)
dimension=ceil(sqrt(initSquare))           cube[ ]=new
array[6][dimension][dimension]          return cube[ ]
end function
```

Listing 4.2 – Function to initialise the Rubik's Cube

4.5.2. Rotation of the Cube

The rotation of the cube requires three arguments – the plane in which the rotation is to be done, the direction of the rotation and the index of the strip that is to be rotated. The algorithm in Listing 4.3 shows an example of the rotation function.

```
function rotate (plane, direction, index)
if (direction = "clockwise")           if
(plane = "Y")
    temp = cube.fetchRow(1, index)
    cube.setRow(1, index, cube.fetchRow(2,
index))    cube.setRow(2, index,
cube.fetchRow(3, index))    cube.setRow(3,
index, cube.fetchRow(4, index))
cube.setRow(4, index, temp)           else if (plane =
"X")
    temp = cube.fetchColumn(1, index)
    cube.setColumn(1, index, cube.fetchColumn(6, index))
cube.setColumnInReverse(6, index, cube.fetchColumn(3, n - index))
    cubesetColumnInReverse(3, n - index, cube.fetchColumn(5,
index))
    cube.setColumn(5, index, temp)
else if (plane = "Z")
    temp = cube.fetchColumn(2, index)
    cube.setColumnInReverse(2, index, cube.fetchRow(6, index))
cube.setRow(6, index, cube.fetchColumn(4, n - index))
cube.setColumnInReverse(4, n - index, cube.fetchRow(5, n - index))
    cube.setRow(5, n - index, temp)
end if
```

```

    else if (direction = "anti-clockwise")
if (plane = "Y")
    temp = cube.fetchRow(1, index)
    cube.setRow(1, index, cube.fetchRow(4,
index))    cube.setRow(4, index,
cube.fetchRow(3, index))    cube.setRow(3,
index, cube.fetchRow(2, index))
cube.setRow(2, index, temp)    else if (plane =
"X")
    temp = cube.fetchColumn(1, index)
    cube.setColumn(1, index, cube.fetchColumn(5, index))
cube.setColumnInReverse(5, index, cube.fetchColumn(3, n - index))
    cube.setColumnInReverse(3, n - index, cube.fetchColumn(6,
index))
    cube.setColumn(6, index, temp)
else if (plane = "Z")
    temp = cube.fetchColumn(2, index)
    cube.setColumnInReverse(2, index, cube.fetchRow(6, index))
cube.setRow(6, index, cube.fetchColumn(4, n - index))
cube.setColumnInReverse(4, n - index, cube.fetchRow(5, n - index))
    cube.setRow(5, n - index, temp)
end if    end if end function

```

Listing 4.3 – Implementation of the rotation function in pseudocode

4.5.3. Preparation of Data for encryption

The system writes the file's byte data onto the face of a virtual customized Rubik's Cube and uses a custom algorithm (Listing 4.3) to create a sequence of rotations to obfuscate the data. Listing 4.4 shows a snippet of code from the method that creates the cube and writes the file data onto its faces.

```

Cube(String fileName){
byte [] data = new byte[0];
try {
    data = FileUtils.readFileToByteArray(new File(fileName));
} catch (IOException e) {
    System.err.println("Could not read file");
    e.printStackTrace();
}
    int ceiling = (int) Math.ceil(data.length /
6.0);    size = (int)
Math.ceil(Math.sqrt(ceiling));    int square = size
* size;
    int totalNumberOfCells = 6 * square;
    data = Arrays.copyOf(data, totalNumberOfCells);
    one = new Face(size, Arrays.copyOfRange(data, 0, square));
two = new Face(size, Arrays.copyOfRange(data, square,
2*square));
    three = new Face(size, Arrays.copyOfRange(data, 2*square,

```

```

3*square));
    four = new Face(size, Arrays.copyOfRange(data, 3*square,
4*square));
    five = new Face(size, Arrays.copyOfRange(data, 4*square,
5*square));
    six = new Face(size, Arrays.copyOfRange(data, 5*square,
6*square));
}

```

Listing 4.4 - Cube constructor code in Java

4.5.4. Encryption Key

The system uses a hash value based on the user's credentials as the encryption key. The system applies a custom algorithm (Listing 4.5) to transform a key string into a sequence of rotations.

1. Take the 128-character hash of the key string.
2. Iterate through all 128 characters.
3. In each iteration, take the integer value of the character.
4. Set the value of the integer from step (3) modulo 3 as the plane in which to perform the rotation.
5. Set $[(x^4 + x^3 + x^2 + x) \bmod \text{size}]$ as the index of the row or column to rotate.

Listing 4.5 – Generation of Rotation Sequence

Listing 4.6 shows a snippet of code from the program which implements the generation of a rotation sequence.

```

private int [][] keyToSequence(String key, int size){
    char [] list;
    list = Hash.getHash(key, "SHA1").toCharArray();

    int [][] sequence = new
int[list.length][2];    int i = 0;    int
plane = 0;    int index = 1;    for (char a :
list){
        sequence[i][plane] = a % 3;
        sequence[i][index] = (a * a * a * a + a * a * a + a * a) %
size;
        i++;
    }
    return sequence;
}

```

Listing 4.6 - Method to convert a key to a rotation sequence

4.5.5. Encryption Function

During an encryption, the rotation sequence is followed forwards and each rotation is carried out in the clockwise direction. Listing 4.7 shows the method which performed the encryption.

```
public void encrypt(String key, String fileName){
    Cube rubik = new Cube(fileName);
    int sequence [][] = keyToSequence(key, rubik.getSize());

    for (int [] rotation : sequence)
        rubik.rotate(rotation[0], rotation[1]);
}
```

Listing 4.7 - Method to encrypt a file using the Rubik's cube

4.5.6. Decryption Function

During a decryption, the rotation sequence is read from the last to the first and each rotation is carried out in the counter-clockwise direction. This undoes the clockwise rotations done during the encryption. Listing 4.8 shows the method which performed the decryption.

```
public void decrypt(String key, String fileName){
    Cube rubik = new Cube(fileName);
    int sequence [][] = keyToSequence(key, rubik.getSize());

    for (int i = sequence.length - 1; i >= 0; i--)
        rubik.reverse(sequence[i][0], sequence[i][1]);
}
```

Listing 4.8 - Method to decrypt a file using the Rubik's cube

4.6. Implementation Of The Data Distribution Module

As stated in Section 3.3 of the methodology, the data distribution module consist of two submodules as File Splitting and Erasure Protection Module (FSEPM), and Shards Dispersal Module (SDM). The FSEPM is implemented using **Reed-Solomon coding** technique and a proposed new **Checksum Data Recovery** technique by this study.

4.7. Reed Solomon Coding

Reed Solomon coding relies on the use of finite field elements known as the Galois Field (GF) to operate. Therefore, in order to make use of Reed Solomon coding for file protection the Galois Field elements, and the Galois Field Arithmetic must be implemented into software so that they can be used for the generation of the Reed Solomon Codeword. The Reed Solomon Codeword is used for the detection and correction of data corruption. Refer to section 2.8 of the literature review for detailed description of the Reed Solomon Coding process. Section 3.3.1 of the methodology presents a

description of how this study uses Reed Solomon Coding. The implementation processes of the GF elements generation, GF arithmetic, and the Reed Solomon Coding are presented in Section 4.7.1 The SMF system's implementation of the GF and the Reed Solomon Coding in JAVA is presented at Section 4.8.

4.7.1. Generating The GF(256) Field Elements

The process of representing a finite field in a computer, especially for arithmetic purposes has been refined. Representing the elements of GF(8) like this {0, 1, 2, 3, 4, 5, 6, 7} for example is more difficult to implement than like this {0, 1, 2, 4, 3, 6, 7, 5}. Doing arithmetic in GF by hand is not much of a problem. However by using a computer the elements of the field are best represented as exponents of 2. An irreducible polynomial is used as a modulus to ensure the exponents of 2 do not repeat.

This example demonstrates the use of the irreducible polynomial 29(i.e. 285-256) to keep the powers of 2 within the range 0 - 255

```

**  $a^0 = 2^0 = 1$ 
**  $a^1 = 1 * 2 = 2$ 
**  $a^2 = 2 * 2 = 4$ 
**  $a^3 = 4 * 2 = 8$ 
**  $a^4 = 8 * 2 = 16$ 
**  $a^5 = 16 * 2 = 32$ 
**  $a^6 = 32 * 2 = 64$ 
**  $a^7 = 64 * 2 = 128$ 
**  $a^8 = 128 * 2 = 256$ 

```

Since 256 is outside the range 0 – 255, we subtract 256 from it to put it back within range, then XOR with the irreducible polynomial (29) to get a unique start value.

Hence $a^8 = (256 - 256) \oplus 29 = 29$

```

**  $a^9 = 29 * 2 = 58$ 
**  $a^{10} = 58 * 2 = 116$ 
**  $a^{11} = 116 * 2 = 232$ 

```

$$** a^{12} = 232 * 2 = 464$$

Once again, 464 is outside the range 0 – 255, so we subtract 256 to put it back within range, then add 29 to get a unique number form what we have previously generated. Hence the calculation continues as:

$$a^{12} = (464 - 256) \oplus 29 = 205$$

$$464$$

$$-256$$

$$208 = 11010000$$

$$\oplus 29 = \quad 11101$$

$$11001101 = 205$$

$$** a^{13} = 205 * 2 = 410 === (410 - 256) \oplus 29 = 135$$

$$** a^{14} = 135 * 2 = 270 === (270 - 256) \oplus 29 = 19$$

$$** a^{15} = 19 * 2 = 38$$

$$** a^{16} = 38 * 2 = 76$$

As can be seen, by repeatedly subtracting 256 from any product value that falls outside of the range of 0 – 255 and XOR with the irreducible polynomial of 29 the process keeps generating unique numbers within the 0 -255 range resulting with the GF(256) table (See Appendix A1).

The pseudocode below describes how to populate two arrays, one to hold the field elements in an order that shows the power of two that they correspond to, and the other to hold the logarithms of the field elements.


```

::function generate_exponent_and_logarithm_arrays() ::
element=1 ::      for i=0 to m-2 ::
exponent[i]=element ::      log[element]=i ::
element=element×2 ::      if element≥field_size
::      element=element XOR prime_number
::      end if ::      end for ::end function

```

Listing 4.9 - Pseudocode for generating the Galois Field elements.

4.7.2. Implementing Arithmetic Operations in GF

Processes involved in the implementation of arithmetic operations in GF as well as pseudocodes are presented below. Java implementations are presented later on in this chapter.

4.7.3. Addition and Subtraction in GF

Additions and subtractions in a Galois Field both come down to the bitwise XOR operation because in GF(2)

$$a. 1 + 1 = 0 \rightarrow 0 - 1 = 1$$

$$b. 1 + 0 = 1 \rightarrow 1 - 1 = 0 \text{ AND } 1 - 0 = 1$$

$$c. 0 + 0 = 0 \rightarrow 0 - 0 = 0$$

Per the definition of the XOR operation, the result is 0 when the operands are alike and 1 when the operands are different. That definition is satisfied by both addition and subtraction in GF(2).

Thus the implementation of the addition and subtraction functions in Java code is simply to XOR the arguments. The pseudocode below describes the function to add field elements. The same function works for subtraction.

```

::function galois_add(x, y)
::      return (x XOR y) ::
end function

```

Listing 4.10 - Pseudocode for performing addition and subtraction in the Galois Field

4.7.4. Multiplication and Division in GF

Multiplication contains an element of addition in it (in fact, multiplication is simply repeated addition) but since addition is implemented as an XOR operation in GF(2), repeated addition will always result in an answer of 0.

An alternative is to use this addition feature of logarithms

$$A * B = \text{antilog}(\log(A) + \log(B));$$

The pseudocode below describes the function to multiply two field elements:

```
::function galois_multiply(a, b) ::      if a=0 OR b=0 ::  
return 0 ::      else ::      return exponent[ ((log  
[a]+log[b]) mod (m-1)) ] ::      end if ::end function
```

Listing 4.11 - Pseudocode for performing multiplication in the Galois Field

Division in the Galois Field is also implemented in Java using logarithms

$$A \div B = \text{antilog}(\log(A) - \log(B));$$

The pseudocode below describes the function to perform division in the finite field.

```
::function galois_divide(a, b) ::      if b=0 ::  
return null ::      else if a=0 ::  
return 0 ::      else ::      return  
exponent[log[a]-log[b]] ::end function
```

Listing 4.12 - Pseudocode for performing division in the Galois Field

4.7.5. Logarithms and Exponents in GF

The precondition for using the addition and subtraction features of logarithms in the multiplication and division in Galois Fields is that the log values for all the elements of the field must be known.

Fortunately the method used to generate the elements of the Galois Field uses exponents of 2.

Thus the log of any element in the field is the exponent of 2 (or exponent of α as in Table 2.2, Chapter 2) indexed to the position corresponding to its generated exponent value. For example, 2^3 generated 8, so $\log(8) = 3$, and 2^7 generated 128, and hence $\log(128) = 7$, in GF(256).

4.7.6. Implementing the algorithm for the generation of coefficients of the encoding polynomial

Both the encoding and decoding processes of the Reed-Solomon algorithm make use of a generator polynomial of the form:

$$g(x) = \prod_{i=1}^n (x - \alpha^i)$$

where n is the number of parity shards.

As noted with Galois Field arithmetic, addition and subtraction both result in the same *XOR* operation. As such, the algorithmic representation of the above formula used the Galois Addition method.

It was determined that the coefficients resulting from the expansion of the encoding polynomial formula

$$g(x) = \sum_{i=0}^n c_i x^{n-i}$$

followed this pattern

$$c_i = \sum_{j=1}^{2t} \prod k$$

where k is a member of one subset of the Galois Field's elements that has i members

1. The algorithm focused on determining the coefficients only. This involved finding all the possible combinations of the first $2t$ Galois Field Elements and performing the appropriate multiplication and addition operations on them. The process used is described in detail below:
2. Obtain the first $2t$ values in the Galois field and save them in an array
3. Determine the highest integer (upper limit) that has $2t$ bits using the formula below

4. $limit = 2^{2t} - 1$
5. Create $2t$ masks for determining which bits are on in any given integer that is within the range $[0, limit]$. The masks are simply integer values whose bit representations have only one bit on and all other bits off. In other words, the masks are the powers of 2 from 2^0 to 2^{2t} . When a bitwise *AND* operation is performed with a number and any of the masks, a result of zero means that the particular bit which the mask has on is off in the number. However, a result greater than zero indicates that that particular bit is on. Using this approach, it is possible both to determine which particular bits are on in the number and also count them.
6. Create $2t$ accumulators, initialized to 0. These accumulators will hold the values of the coefficients when the program runs to a completion.
7. The combinations of the first $2t$ Galois Field elements are generated as integer values from 1 to $limit$. Thus, a loop is used which runs from 1 to $limit$. For each integer in the range a . The masks are applied to the integer to determine which bits are on.
 - b. The Galois Field elements, whose positions correspond to the bits that are on in the integer, are multiplied to get the product of all the elements in that particular combination.
 - c. Concurrently, the number of bits that are on are counted. The product from multiplying the bits that are on is added to the accumulator at the position of the count. This ensures that all the products derived from multiplying combinations of a particular length are accumulated in a single accumulator.

The task of generating the encoding polynomial can be quickly and efficiently carried out using a loop that continually multiplies the next monomial in the $g(x)$ sequence by the previously obtained polynomial. Two functions are required to generate the encoding polynomial.

1. Polynomial Multiplication Function: this function implements polynomial multiplication in code.
2. Encoding Polynomial Function: this function makes use of polynomial multiplication to generate the encoding polynomial using iteration.

4.7.7. Polynomial Multiplication

Polynomial multiplication is based on the distributive property of multiplication over addition, and the multiplicative law of indices. An example is shown below, showing the steps in finding the product of $5x^2 + 8x + 3$ and $3x + 8$.

$$(5x^2 + 8x + 3) \times (3x + 8)$$

Distribute the elements of the first polynomial over the second one

$$= 5x^2(3x + 8) + 8x(3x + 8) + 3(3x + 8)$$

Multiply each element of the first polynomial by each element of the second

$$\begin{aligned} &= (5x^2 \times 3x) + (5x^2 \times 8) + (8x \times 3x) + (8x \times 8) + (3 \times 3x) + (3 \times 8) \\ &= 15x^3 + 8x^2 + 24x^2 \end{aligned}$$

Sum the terms of the generated polynomial which have the same degree of x

$$= 15x^3 + 32x^2 + 73x + 24$$

The following pseudocode describes a function to multiply two polynomials. The coefficients of the polynomials are stored in an array with the array index describing the degree of that term in the polynomial.

```
::function polynomialMultiplication(a[ ], b[ ])
::      for i=0 to length of a
::          for j=0 to length of b
::              product[i+j]=product[i+j]+(a[i]*b[j])
::          end for
::      end for
::end function
```

Listing 4.13 - Pseudocode for polynomial multiplication

4.7.8. The Encoding Polynomial Function

The encoding polynomial is the product of the polynomials of the form $(x - 2^i)$ for i values from 1 to the number of parity data.

The pseudocode below describes a function used to generate the encoding polynomial for n number of parity. The encoding polynomial function simply loops n times, generating a new monomial and multiplying it by the previously obtained encoding polynomial. The encoding polynomial is initialized to $\{1\}$.

```
::function generate_encoding_polynomial( n)
::      polynomial[ ]={1}
```

```

::   for i=1 to n
::       polynomial=polynomial_multiplication(polynomial, {2^i, 1})
::   end for
::   return polynomial
::end function

```

Listing 4.14 - Pseudocode to generate the Reed Solomon Encoding Polynomial

4.7.9. Reed Solomon Codeword

One simple way to treat a message as a polynomial is to read the message into an array. The index of the array at which an element is found, is used as the power of x that the element multiplies in the polynomial. For instance, the array

$$M = \{2, 3, 4, 6, 5, 1\}$$

Could easily represent the polynomial

$$M(x) = x^5 + 5x^4 + 6x^3 + 4x^2 + 3x + 2$$

The pseudocode below describes a function for generating the Reed Solomon Codeword from a given message polynomial $M(x)$, using parity value of n .

```

::function encode_polynomial(M[ ], n)
::   encoding_polynomial[ ]=generate_encoding_polynomial(n)
::   M=polynomial_shift(M, n)
::   quotient, remainder=polynomialDivision(M, encoding_polynomial)
::   codeword=polynomialAddition(M, remainder)
::   return codeword
::end function

```

Listing 4.15 - Pseudocode for encoding a polynomial using an RS Encoding Polynomial

4.8. Implementation Of File Splitting And Erasure Protection Module (FSEPM) Using Java

For the SMF system to guard against data loss or corruption, it makes use of Reed Solomon coding or the checksum data recovery technique to create parity data with which the file can be reconstructed in the event of data loss or corruption. The SMF system user is given the choice of choosing a file priority at the time of uploading a file to the cloud. The selected file priority determines the parameters for

creating the parity data. The system provides four file priority levels as follows; “**Low**”, “**Normal**”, “**Important**” and “**Critical**.”

4.8.1. Implementation of the Erasure Protection via Reed Solomon Coding

Reed Solomon Coding refers to a method of error detection and correction that computes recovery information before the file is transmitted. The recovery information, called “parity”, is transmitted together with the file data. The presence of an “error” can be detected by examining the parity information while an “erasure” is the complete absence of a portion of the data. To perform a Reed Solomon Encoding, the system needs the number of data shards as well as the number of parity shards. These numbers are passed to the encoding method which breaks the file into “data” number of shards and computes “parity” number of metadata information which can be used to recover lost or corrupt data shards.

The system receives a file priority setting from the user at the time of selecting a file for upload. Based on the file priority setting, the system determines the number of data and parity shards to use.

The system can recover up to “parity” number of shard loss and can correct up to half “parity” number of shard corruption. This means that the larger the number of parity data, the more likely it is that the file can be reconstructed in case there is some loss of data. Hence, files with a higher priority are given a larger number of parity. This makes recovering them more likely than recovering files with a lower priority. However, the higher the number of parity shards, the more the computation that goes into checking for errors and recovering missing or corrupt files. As such, files with higher priority take a longer time to reconstruct and also require more memory for the reconstruction operation. They also require more disk space locally and on the cloud.

The priority levels that use Reed Solomon coding and their associated data and parity shard counts are specified below. In all cases, SecureMyFiles breaks the file into a number of data shards and computes the parity shards so that the total number of shards is 144. Any number from 2 to 256 can be used for the process, but 144 is an optimal value because it balances the computation time and strength of the Reed Solomon Encoding/Decoding process. In other words, using a total shard count of 256 would have been the most secure implementation of Reed Solomon encoding/decoding but that would likely make heavy use of the device’s CPU and memory. Further, SMF prefers that the minimum number of CSPs connected to the SMF client is 6. Listing 4.16 presents a snippet of code that shows how the data and parity shard counts are set based on the user’s choice of a priority setting (i.e. Low, Normal, or Important).

```

if
(filePriority.equalsIgnoreCase("low")){
dataShards = 120;    parityShards = 24;
}
else if
(filePriority.equalsIgnoreCase("normal")){
dataShards = 96;    parityShards = 48;
}
else if
(filePriority.equalsIgnoreCase("important")){
dataShards = 72;    parityShards = 72;
}
}

```

Listing 4.16 - Snippet of code to show how the data and parity shard counts are set based on the priority setting

Low - Files with “low” priority are split into 120 data shards and 24 parity shards, regardless of the size of the file. In other words, the splitting has no bearing on the size of the file. Each of the 6 Cloud Service Providers receives 24 shards for storage. This offers the least protection since this configuration allows for 12 error correction and 24 erasure recovery, meaning that the subscriber can recover data if one of the service providers is not available. However, the computation is fastest and uses less memory.

Normal - Files with “normal” priority are split into 96 data shards and 48 parity shards. Therefore, 24 errors can be corrected and 48 erasures can be recovered. This means that the data can be recovered even if two of the subscriber’s Cloud Service Provider are unavailable. However, the encoding operation is slower than with the “low” priority files and it requires more memory.

Important - Files with “important” priority are split into 72 data shards and 72 parity shards, allowing 36 errors to be corrected and 72 erasure recoveries. In this configuration, the data can be recovered even if three CSPs are unavailable. This uses the most computational power as well as memory.

4.8.2. Implementation of the Reed Solomon Encoding Process

The Reed Solomon Encoding processes outlined in section 4.7 are implemented using JAVA as follows:

Generation of the Galois Field Elements - The Reed Solomon encoding process represents data as integers in a Galois Field. The SMF system employs a field of size 256 and an irreducible polynomial of 285 (i.e. 256 plus the prime number 29). The Galois Field comprised of the values for exponents of two, computed from the power 0 to the power 255, and zero. Any time a power of two exceeds the field size the value is “added” to the irreducible polynomial to produce a number that is within the field (Refer to section 2.8.3). All the powers of two are stored in an array for easy access when they are needed for multiplication and division operations. Also, the exponents that yield a power value are also stored in a separate array for easy access in multiplication and division operations as shown in Listing 4.17.

```
private void generateGaloisField(){      byte alpha = 1;      log =
new byte[size];      exp = new byte[size * 2 - 2];      for (int i =
0; i < size - 1; i++) {      exp[i] = exp[i + size - 1] = alpha;
log[alpha] = (byte) i;      alpha = (byte) (alpha << 1);
if ((alpha & 0xFF) >= size)      alpha ^= polynomial;
    }
    log[0] = -1;
}
```

Listing 4.17 – Generation of the Galois Field Elements in Java

4.8.3. Galois Field Arithmetic

Addition – Addition in the Galois Field is implemented as a bitwise XOR operation (Listing 4.18).

```
public short add(byte a, byte b){      return (byte) (a
^ b);
}
```

Listing 4.18 - Implementation of addition in the Galois Field

Multiplication – Multiplication is carried out by taking the antilog of the sum of the logarithms of operands (Listing 4.19).

```
Public short multiply(byte a, byte b){      if (a == 0
|| b == 0) return 0;      return exp[log[a] + log[b]];
}
```

Listing 4.19 - Implementation of multiplication in the Galois Field

Division – Division is carried out by multiplying the dividend by the inverse of the divisor (Listing 4.20).


```

short divide(byte a, byte b){
if (b == 0)
    throw new IllegalArgumentException("Division by zero is not
allowed");

    return multiply(a, inverse(b)); }

```

Listing 4.20 - Implementation of division in the Galois Field

Logarithm – Logarithms are taken from the array created when the Galois Field is generated. They are the exponents of two that generates the field values (Listing 4.21).

```

public short getLog(int i) { return log[i]; }

```

Listing 4.21 - Method for accessing the log of a field element

Antilog(Exponents) – Exponents are taken from the array created when the Galois Field is generated (Listing 4.22).

```

public short getExp(int i) { return exp[i]; }

```

Listing 4.22 - Method for accessing the exponent of two in the Galois Field

4.8.4. Polynomial Arithmetic

The Reed Solomon encoding process treats strands of data as polynomials. The system uses an array to represent a polynomial with the index of the array element corresponding to its degree (Listing 4.23).

```

Polynomial(byte [] data){    this.data = data;
}

```

Listing 4.23 - Polynomial class constructor

Four methods are used to implement the addition, multiplication, division and modulus operations.

Addition – Addition is carried out by summing the values of the operand arrays which have the same index (Listing 4.24).

```

private Polynomial plus(Polynomial poly){      short [] a
= (this.getDegree() >= poly.getDegree()) ?
this.getData() : poly.getData();      short [] b =
(this.getDegree() < poly.getDegree()) ?
this.getData() : poly.getData();

    for (int i = 0; i < b.length; i++)
a[i] ^= b[i];

    return new Polynomial(a); }

```

Listing 4.24 - Implementation of Polynomial Addition

Multiplication – The polynomial multiplication method makes use of the distributive property of multiplication over addition, returning a polynomial whose length is the sum of the lengths of the two operands (Listing 4.25).

```

Polynomial times(Polynomial poly, Galois
g){      short [] a = this.getData();
short [] b = poly.getData();

    short [] product = new short[this.getDegree() +
poly.getDegree() + 1];
    Arrays.fill(product, (short) 0);

    for(int i = 0; i < a.length; i++)
for (int j = 0; j < b.length; j++)
        product[i + j] ^= g.multiply(a[i], b[j]);

    return new Polynomial(product).trim();
}

```

Listing 4.25 - Implementation of Polynomial Multiplication

Division – The polynomial division method followed the principles of polynomial division in algebra. The method returns the quotient of the division (Listing 4.26).

```

    Polynomial dividedBy(Polynomial poly, Galois g){        if
        (this.getDegree() < poly.trim().getDegree() ||
            this.isZero())

        return new Polynomial(new short[]{0});

    short [] a = reverseOf(this.getData());
    short [] b = reverseOf(poly.getData());

    short [] q = new short[this.getDegree() - poly.getDegree()+ 1];
    for (int i = 0; i < q.length; i++) {        q[i] = g.divide(a[i],
b[0]);        for (int j = 0; j < b.length; j++)        a[j +
i] ^= g.multiply(q[i], b[j]);
    }
    return new Polynomial(reverseOf(q)).trim();
}

```

Listing 4.26 - Implementation of Polynomial Division

Modulus – The polynomial modulus method uses the same approach as the polynomial division but returns the remainder instead of the quotient (Listing 4.27).

```

Polynomial mod(Polynomial poly, Galois g){
    if (this.getDegree() < poly.getDegree() || this.isZero())
return new Polynomial(this.getData());

    short [] a = reverseOf(this.getData());
    short [] b = reverseOf(poly.getData());

    int l = this.getDegree() - poly.getDegree() + 1;
short q;
    for (int i = 0; i < l; i++) {        q
= g.divide(a[i], b[0]);        for (int j
= 0; j < b.length; j++)        a[j +
i] ^= g.multiply(q, b[j]);
    }
    short [] remainder = new short[a.length - 1];
    System.arraycopy(a, l, remainder, 0, remainder.length);
return new Polynomial(reverseOf(remainder)); }

```

Listing 4.27 - Implementation of the Polynomial Modulus

4.8.5. Generator Polynomial

Reed Solomon encoding and decoding both make use of a special polynomial called the Generator Polynomial. The generator polynomial is generated by performing a polynomial multiplication of a

series of monomials whose constant terms are the sequence of powers of two from 1 to two to the power of “parity”, and the first-degree terms all have the coefficient 1 (Listing 4.28).

```
encodingPolynomial = new Polynomial(new short[]{1}); for (int i = 1;
i <= paritySize; i++){
    encodingPolynomial = encodingPolynomial.times(new Polynomial(new
short[]{galois.getExp(i), 1}), galois);
}
```

Listing 4.28 - Code for generating the Encoding Polynomial

4.8.6. File Encoding

The selected file is first read into a two-dimensional byte array. The number of rows in the array corresponded with the number of data shards that is set according to the file priority. The system iterates through the columns of the array and extracts each column's data into a polynomial. The system then computes the parity data as the remainder when the data polynomial is divided by the generator polynomial. The parity data from all the columns are joined into their own twodimensional array to form the parity data for the entire file. Finally, each row from the data array and the parity array are read into separate files, thus creating shards (pieces) of the file and parity (Figure 4.3). Listing 4.29 is the code snippet of the file encoding process using Reed-Solomon Coding.

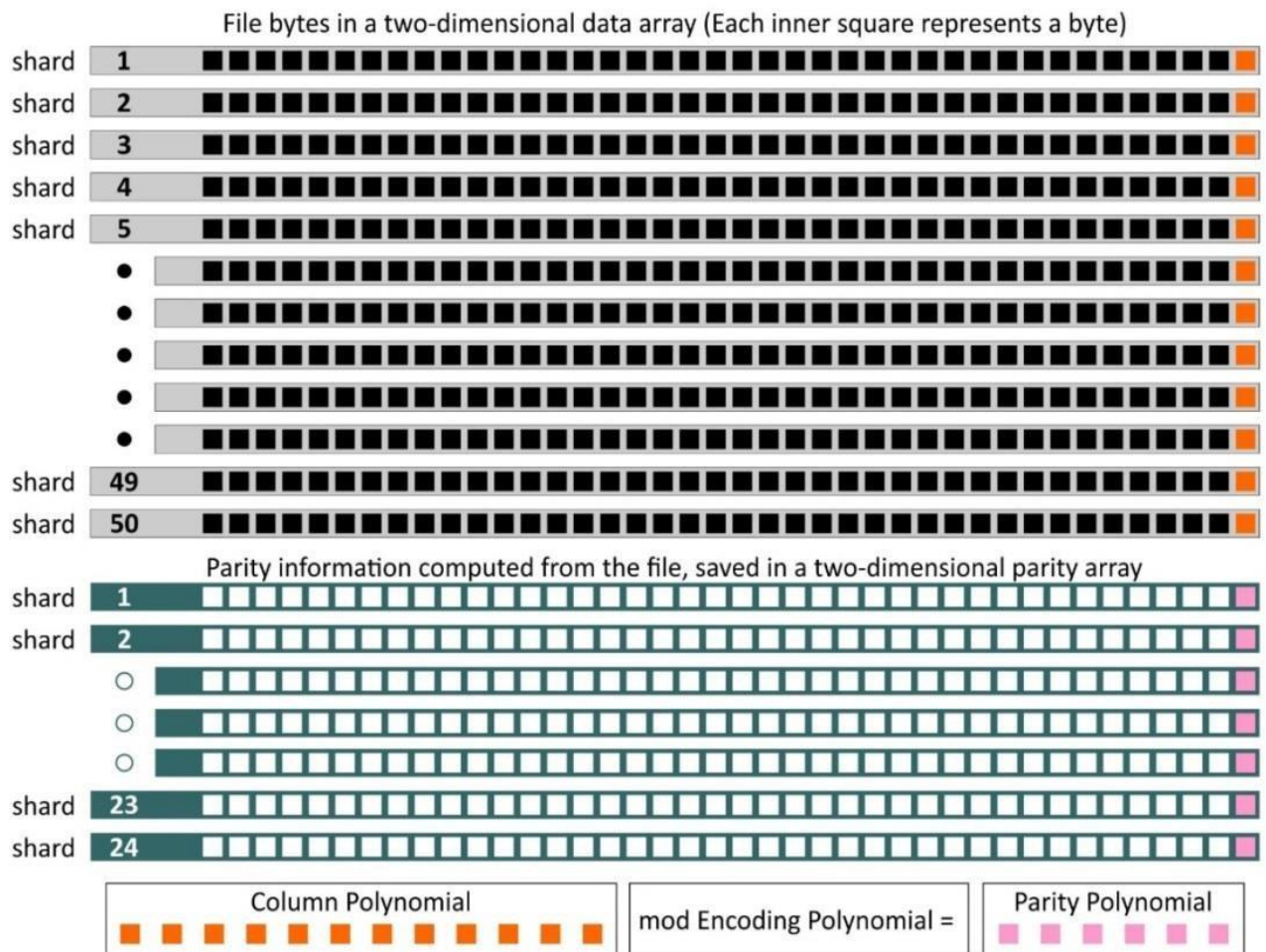


Figure 4.3 - Splitting of file and computation of parity.

```
public int encode(String fileName,
int dataSize,
int
paritySize){

    ReedSolomon rs = new ReedSolomon(256, 29, dataSize,
paritySize);
    FileUtility fileUtility = new FileUtility();

    short [][] file = fileUtility.fileToShort(fileName,
dataSize);    if (file == null)        return -1;

    short [][] output = new short[dataSize +
paritySize][file[0].length];
    for (int i = 0; i< output.length; i++){
```

```

        Arrays.fill(output[i], (short) 0);
    }
    for (int i = 0; i < file[0].length; i++){
        Polynomial column = new
            Polynomial(fileUtility.extractColumn(file, i));
        column = rs.encode(column);
        fileUtility.attachColumn(output, column.getData(), i);
    }
    fileUtility.shortToFile(output, fileName);

    return output[0].length; }

```

Listing 4.29 - Java implementation of file encoding process using Reed-Solomon Coding

4.9. Implementation Of The Checksum Data Recovery Technique

Unlike the other file priority options, the fourth priority level of the SMF system dubbed “**critical**” does not use Reed Solomon coding to protect the data. Instead it employs this study proposed **Checksum Data Recovery** technique (Refer to Chapter 3, Section 3.3.2) that computes checksums for a file by taken bytes from several different sets of data bytes. The checksum information is stored in a metadata server for future reference in order to ascertain whether the user’s data has been corrupted. Furthermore, the checksum data is used to correct errors within the file.

4.9.1. Encoding

The processes involved in using the proposed Checksum Data Recovery technique are presented below.

4.9.2. Internal Data Representation

- The system first reads data into a three-dimensional array. The array has a special property that the cross section is 4×4 array that represents a single module (a data shard). The pseudocode in Listing 4.30 represents a method to perform the conversion of the file data into a three-dimensional array. Figure 4.4 illustrate the Operations of the Checksum Data Recovery (CDR). It depicts the formation of shards from the modules of the CDR, using a 512 byte file.

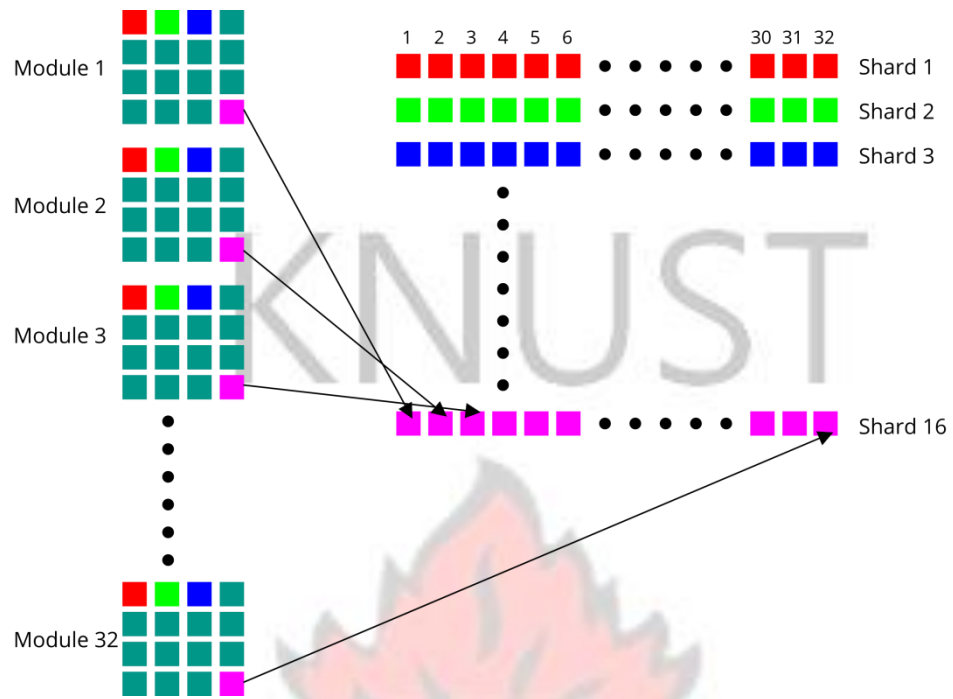


Figure 4.4: The formation of shards from the modules of the CDR using a 512 byte file.

```
function readFileToArray(file)    oneDimensionalArray =
readFileToByteArray(file)    breadth =
ceiling(oneDimensionalArray.length / 16.0)
    threeDimensionalArray = new array[breadth][4][4]
    a = 0
    for i=0 to breadth
        for j=0 to 4
            for k=0 to 4
                threeDimensionalArray[i][j][k] =
                    oneDimensionalArray[a]
                a++
            endfor
        endfor
    endfor
endfunction
```

Listing 4.30 – Function to read from a file into a three-dimensional array

To implement above algorithm, the system read data from a file into a one-dimensional array using a method from the Apache Commons IO library. The system then writes the data into a three-dimensional array in order to get modules for the computation of parity information. Listing 4.31 shows a snippet of the code used to perform the data reading and conversion.

```
public Data(String fileName) throws Exception {
    this.filename = fileName;
    byte [] fileAsByteArray = FileUtils.readFileToByteArray(new
    File(fileName));
    int i=0;      int
    four = 4;
    int breadth = (int) Math.ceil(fileAsByteArray.length / 16.0 );

    array = new byte[breadth][four][four];
    for (int j=0; j<breadth; j++){      for (int
    k = 0; k < four; k++){      for(int l=0;
    l < four; l++){      if (i <
    fileAsByteArray.length){
        array[j][k][l] = fileAsByteArray[i++];
    }
    else {
        array[j][k][l] = 1;
    }
    i++;
    }
    }
}
}
```

Listing 4.31 – reading file to 3-dimensional array

4.9.3. Parity Data Computation

Module Parity Computation - After reading data into a three-dimensional array, the system computes the parity value for each 4×4 module of the three-dimensional array. The pseudocode in Listing 4.32 represents a function to compute the module parity.

```
function computeModuleParity()
    sum = 0    for row = 0 to 4        for
    column = 0 to 4        sum = sum XOR
    module[row][column]
    endfor
    endfor
    return sum
endfunction
```

Listing 4.32 – Function to compute the module's parity

Row and Column Parity - The system proceeds to compute the parity values for each row and each column, but instead of simply summing all the values in each row and column, the system sums all the different combinations of the row data as well as the different combinations of the column data as depicted by Figure 3.7 - Charter 3. The code snippet of Listing 4.33 represents a function that computes the row and column parities.

```
for(int module = 0; module < this.data.length; module++){
    this.moduleParity[module] =
    ComputeModuleSum(this.data[module]);
    int x =0, b = 0;
    for(int i =0; i < this.data[module].length; i++){
        for(int j =0; j < this.data[module][i].length; j++){
            this.rowsParity[module][i][6] ^=
            this.data[module][i][j];
            for(int k = j+1; k < this.data[module][i].length;
            k++){
                this.colsParity[module][x][i] =
                (byte) (this.data[module][j][i] ^ this.data[module][k][i]);
                x++;
            }
            for(int l = j+1; l < this.data[module][i].length;
            l++){
                this.rowsParity[module][i][b] =
                (byte) (this.data[module][i][j] ^ data[module][i][l]);
                b++;
            }
        }
        x = 0;
        b = 0;
    }
    this.colsParity[module][6] = this.data[module][3];
}
```

Listing 4.33 – Function to compute row and column parities

The system computes the parity data for each module as well as the rows and columns in the modules. The parity for a module is computed as the XOR sum of all the elements in the 4×4 grid that made up the module and is stored in a single-dimensional array.

The code snippet in Listing 4.34 shows the implementation of the module parity computation.

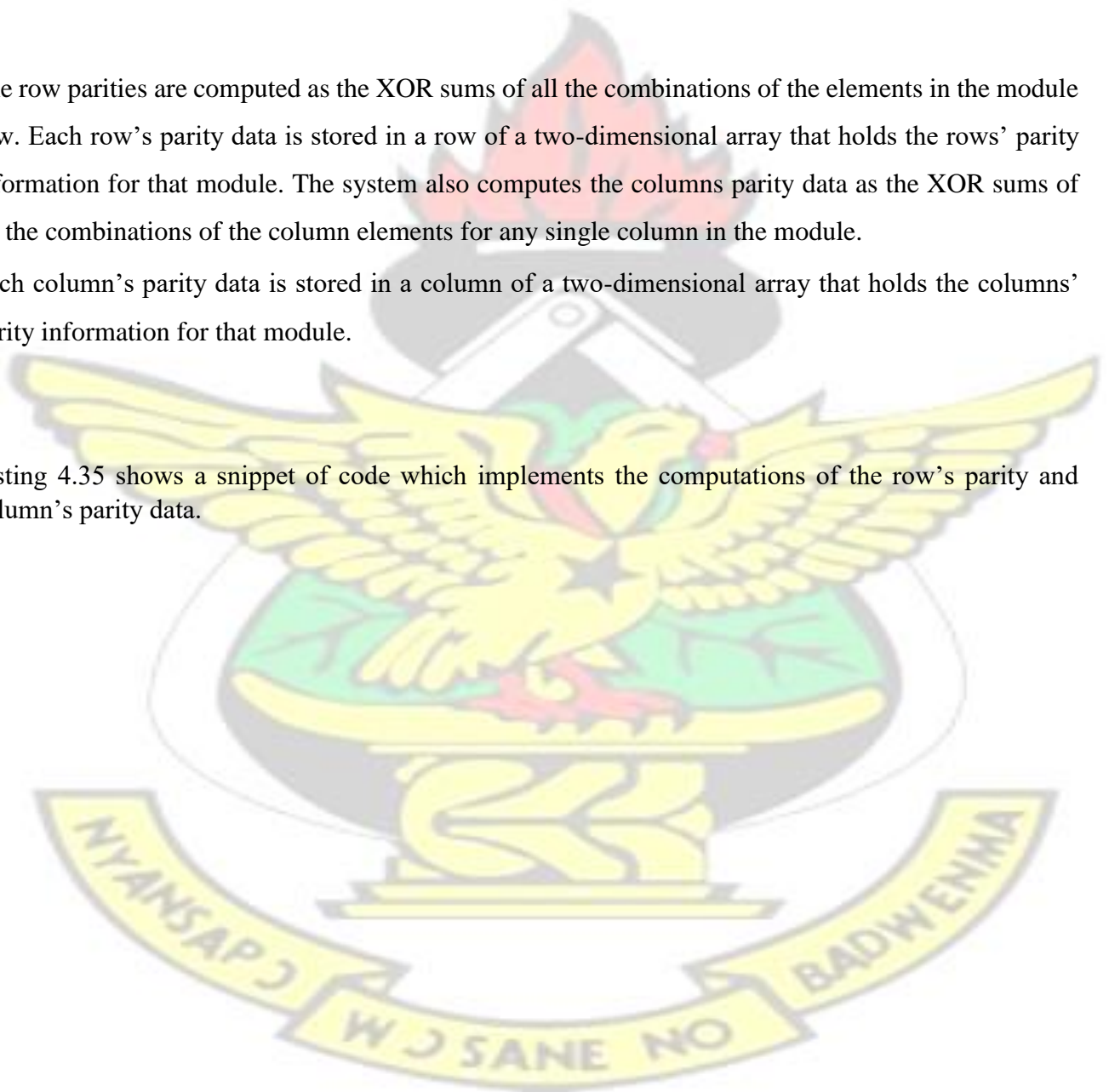

```
private byte ComputeModuleSum(byte[][] data){
byte sum = 0;
    for(int row = 0; row < data.length; row++){
        for(int col = 0; col < data[row].length; col++){
            sum ^= data[row][col];
        }
    }
    return sum;
}
```

Listing 4.34 – Computation of module parity

The row parities are computed as the XOR sums of all the combinations of the elements in the module row. Each row's parity data is stored in a row of a two-dimensional array that holds the rows' parity information for that module. The system also computes the columns parity data as the XOR sums of all the combinations of the column elements for any single column in the module.

Each column's parity data is stored in a column of a two-dimensional array that holds the columns' parity information for that module.

Listing 4.35 shows a snippet of code which implements the computations of the row's parity and column's parity data.



```

        for(int module = 0; module < this.data.length; module++){
            this.moduleParity[module] =
ComputeModuleSum(this.data[module]);
            int x =0, b = 0;
            for(int i =0; i < this.data[module].length; i++){
                for(int j =0; j < this.data[module][i].length; j++){
                    this.rowsParity[module][i][6] ^=
this.data[module][i][j];
                    for(int k = j+1; k <
this.data[module][i].length; k++){
                        this.colsParity[module][x][i] =
(byte) (this.data[module][j][i] ^ this.data[module][k][i]);
                        x++;
                    }
                    for(int l = j+1; l <
this.data[module][i].length; l++){
                        this.rowsParity[module][i][b] =
(byte) (this.data[module][i][j] ^ data[module][i][l]);
                        b++;
                    }
                }
            }
            x = 0;
            b = 0;
        }
        this.colsParity[module][6] = this.data[module][3];    }

```

Listing 4.35 – Computation of row parity and column parity

4.9.4. Writing Parity Data to File

The system creates a parity object from the computed parity information and writes the data to three files for the module parity, row parity and column parity. The code snippets in Listing 4.36 and Listing 4.37 respectively shows implementations of how the parity objects are populated and written to file.

```

Parity p = new Parity();    p.setColumn(colsParity);
p.setRow(rowsParity);
p.setModule(moduleParity);

```

Listing 4.36 – Population of the parity object

```

public void writeToFile(String fileName) throws IOException
{
    int length = row.length * 28;
    byte[] array = new
byte[length];

    File dir = new File(DIR);
    if (!dir.exists()) dir.mkdir();

    fileName = DIR + "/" + fileName;

    FileUtils.writeByteArrayToFile(new File(fileName + ".module"),
module);

    int i = 0;
    for (byte[][] a : row){
for (byte[] b : a){
for (byte c : b){
array[i++] = c;
        }
    }
}

    FileUtils.writeByteArrayToFile(new File(fileName +
".row"), array);
    i = 0;
    for (byte[][] a : column){
for (byte[] b : a){
for (byte c : b){
array[i++] = c;
        }
    }
}

    FileUtils.writeByteArrayToFile(new File(fileName + ".col"),
array);
}

```

Listing 4.37 – Writing parity data to file

4.9.5. Data Splitting

After the parity data has been computed for the file data, the system splits the file into 16 shards. Each shard comprises of all the module elements of a particular location in the 4×4 grid. That is, every first element in the 4×4 grid is collected into one shard. The algorithm in Listing 4.38 represents a function to split the data into shards for uploading to the cloud.


```

function splitData()      splits = new
array[16][threeDimensionalArray.length]      for i = 0
to 4          for j = 0 to 4          current =
(4 * i) + j
          for k = 0 to threeDimensionalArray.length
            splits[current][k] =
threeDimensionalArray[k][i][j]
          endfor
          writeByteArrayToFile(splits[current])
        end
      for      end
    for end function

```

Listing 4.38 – Function to split file into shards

The algorithm is implemented using the Apache Commons IO Java library (Listing 4.39).

```

public void split(Data data){
    String dir = "temp/split";
    File directory = new File(dir);
    if (!directory.exists()) directory.mkdirs();

    byte [][] splits = new
byte[16][data.length];  for (int i = 0; i < 4;
i++){
        for (int j = 0; j < 4; j++){
            int a = 4 * i + j;
            for (int k = 0; k < data.length; j++){
                splits[a][k] = data[k][i][j];
            }
            FileUtils.writeByteArrayToFile(splits[a], new
File(dir + "/" + data.fileName + "." + a));
        }
    }
}

```

Listing 4.39 – Splitting data into 16 shards

4.9.6. Checksum Data Recovery Technique Metadata

There are three types of metadata, the rows parities metadata, columns parities metadata, and module sum parity.

Module sum parity is a single dimensional array of length similar to the length of the 3D data array.

The entry values are computed from the XOR of all entries of a module and stored in the array.

The module sum parity array has two main benefits:

- It ensures no module is dependent on another module (i.e. module abstraction).
- It reduces unnecessary iterations. Thus, the program checks for corrupted entries in a module only when the sum of entries in the module is not equal to the value of module sum parity entry at that index.

Each of rows parities and columns parities metadata is a 3D array of size similar to the 3D data array computed from the data file. Each entry in the metadata array is a 2D array. Which implies the 3D metadata arrays also contains modules in its entries.

A module in the rows parities array is a $4 \times 4C2+1$ matrix and the columns parities array is a $4C2+1 \times 4$ matrix, where C is combination. Thus, for a byte of data to be uniquely recovered in the event of an error, each byte in a module must have relationships with entries on the same row and also entries on the same column. For example, for a module of size 4×4 matrix, entry $a(0,0)$ is related to entries $a(0,1)$, $a(0,2)$ and $a(0,3)$ as these are entries are on the same row. These relationships are stored as row parities metadata. Similarly, entry $a(0,0)$ is also related to entries $a(1,0)$, $a(2,0)$ and $a(3,0)$ as these are entries on the same column. These relationships are also stored as columns parities metadata.

Clearly there is a relationship between any two entries on the same row or column as there are four entries on each row and four on each column of a module. The parity metadata for a given file is obtained by computing the relationships on each row as 4 combination 2 (i.e. taking 2 entries at a time). The combination (not permutation) used here is important because the order of combining is not important since the relationship is computed as the XOR of each pair of entry and this minimises the size of the metadata arrays.

The additional column on the row parities array holds the XOR of all entries in a row. This column is very important because the algorithm uses the value held in the column to recover corrupted entries on the row when the sum of the row does not equate to the value held.

The additional row in a column parities array module contains the AND of the last row and itself. This is important because the whole error detection algorithm will use the last row as its base or reference point.

The possible pairing is illustrated in Chapter 3, Figure 3.8.

4.9.7. How A Module Locates Its Metadata In The Proposed Checksum Data Recovery Technique

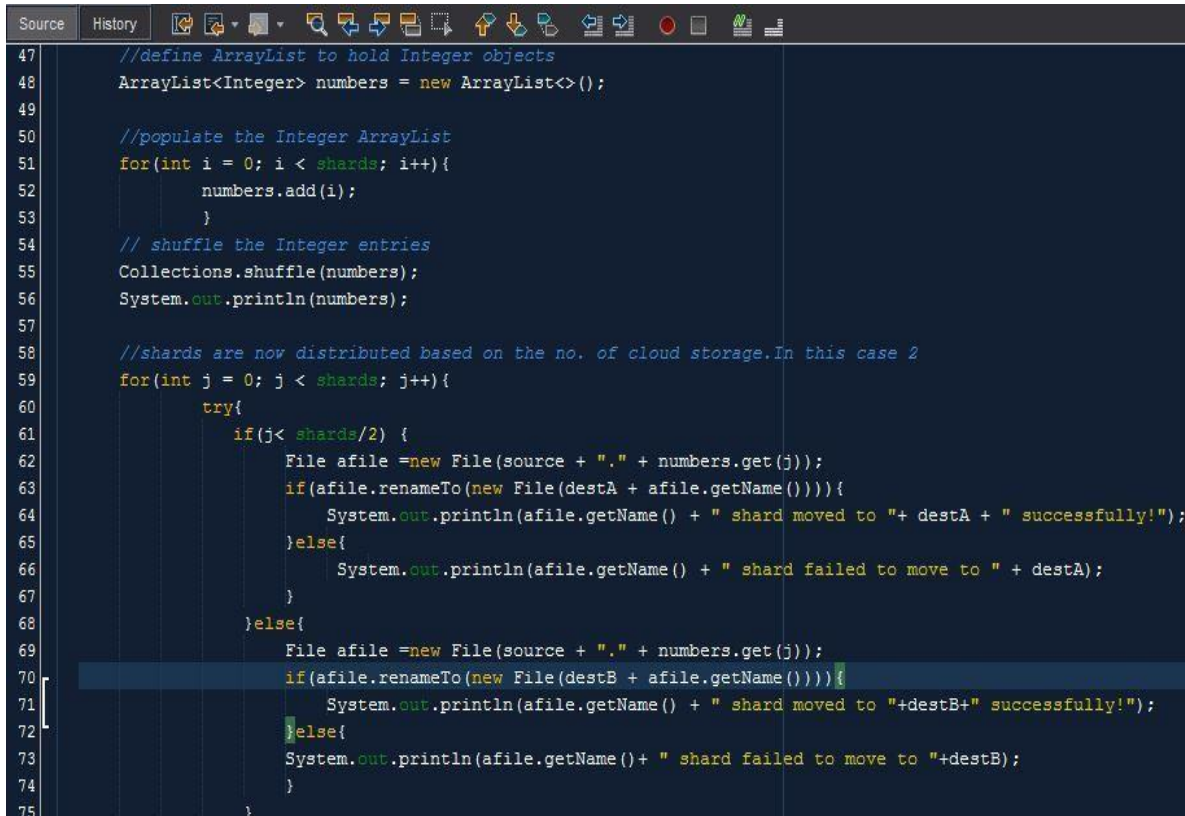
Given a module M [4] [4] at index [x] [4] [4] in the 3D data array,

- Its corresponding module sum parity is at index [x] within the module sum parity metadata array (a single dimensional).
- Its rows parity module is at index [x] of the rows parity metadata array.
- Its columns parity module is at index [x] of the columns parity metadata array.

4.10. Implementation Of The Data Dispersal Technique (The Shuffling Method)

The shuffling is implemented using the Java Collection class in the Utility package. An array list is created to contain integer numbers that correspond to the number of the derived shards after splitting of file. The shuffling of the integer numbers in the array list is achieved by passing the array list as a parameter to the static Shuffle method called from the Collections class. Appending the numbers from the newly shuffled list to the original file name, a new name is formed (filename.extension.number) which is used to get the individual shards from the source (temporary storage where the shards are kept after splitting) to be ready for upload. Now the data shards are distributed depending on the SMF user's selected number of cloud providers. For instance, if two cloud storage providers are selected (i.e. Dropbox and Box), then half of the appended list are distributed to the Dropbox and the other half to the Box.

Below is the snippet code of the implementation of the data dispersion technique (Listing 4.40).



```

47 //define ArrayList to hold Integer objects
48 ArrayList<Integer> numbers = new ArrayList<>();
49
50 //populate the Integer ArrayList
51 for(int i = 0; i < shards; i++){
52     numbers.add(i);
53 }
54 // shuffle the Integer entries
55 Collections.shuffle(numbers);
56 System.out.println(numbers);
57
58 //shards are now distributed based on the no. of cloud storage. In this case 2
59 for(int j = 0; j < shards; j++){
60     try{
61         if(j < shards/2) {
62             File afile = new File(source + "." + numbers.get(j));
63             if(afile.renameTo(new File(destA + afile.getName()))){
64                 System.out.println(afile.getName() + " shard moved to " + destA + " successfully!");
65             }else{
66                 System.out.println(afile.getName() + " shard failed to move to " + destA);
67             }
68         }else{
69             File afile = new File(source + "." + numbers.get(j));
70             if(afile.renameTo(new File(destB + afile.getName()))){
71                 System.out.println(afile.getName() + " shard moved to " + destB + " successfully!");
72             }else{
73                 System.out.println(afile.getName() + " shard failed to move to " + destB);
74             }
75         }
76     }
77 }

```

Listing 4.40 - Code for implementation of the data dispersion technique (the shuffling method)

4.11. Implementation Of SMF System's Metadata

In order to keep track of file information, the system uses a metadata file (dubbed, *file metadata*) to record information about the file and its shards. The information the system captures includes:

1. File Name
2. File Size
3. File Priority
4. Date of File Upload
5. Number of Data Shards
6. Number of Parity Shards
7. Cloud Service Destinations for the Shards

The system hosts the *file metadata* on SMF servers for easy access from multiple devices and locations. A plain text file is used to hold the metadata information. The data is encoded in **JavaScript Object Notation (JSON)**. The **Google Gson library** is used for encoding and decoding the metadata.

After a file is selected for upload and the file priority is set, a metadata object is created and the fields in the object are set with the file name, size, priority, number of data and parity shards. After the shards are shuffled and assigned cloud accounts, the destination information is also captured into the metadata object (Listing 4.41).

The system uses the Gson library to create a JSON string from the metadata object. The string is then written to a file with the hash of the original file name as the name of the metadata file (Listing 4.42). The extension for metadata files is “.meta”. When the system is done creating and updating the contents of the metadata file, the file is uploaded to the SMF Metadata Server for safekeeping.

```
metadata.File metadata =  
    new metadata.File(tempFile.getName(),  
Date.from(Instant.now()),      filePriority,  
dataShards,                    parityShards,      cols,  
                                size);
```

Listing 4.41 - Instantiation of a metadata file object

```
public static void writeFile(String content, File file) {  
    FileWriter writer = new  
FileWriter(file);    writer.write(content);  
writer.close();  
}
```

Listing 4.42 - Method to write metadata string to file

4.12. Shards Upload Module

After the shards have been prepared for upload, SMF makes use of the Application Programmer Interface (API) of the Cloud Service Provider to send the file over the internet to the CSP. Listing 4.43 shows a snippet of code from the SMF that does the file upload to Dropbox and Box.

```
if (cloud.equalsIgnoreCase("dropbox"))  
    try {  
        dbx.uploadFile(f);  
    } catch (IOException | DbxException e) {  
        e.printStackTrace();  
    }
```

```

    }
else if (cloud.equalsIgnoreCase("box"))
    try {
        box.uploadFile(f);
    } catch (IOException e) {
        e.printStackTrace();
    }
}

```

Listing 4.43 - Snippet of code showing file upload to Dropbox and Bo

4.13. File Download Module

The download module is composed of seven (7) activities as shown by Figure 4.4.

The user selects the file to download from their profile by clicking on the appropriate file name which is actually kept in the *user metadata*.

The selected file name is **hashed** by applying *the same hash function* as that of the upload module to obtain the same hashed value if the file integrity has not been tempered with; the resulting hashed value is used with the file metadata file to obtain the required split chunks or shards from the user's cloud accounts storing them to a temporary storage (a **buffer area**).

The downloaded shards are then decoded or joined through using the **Reed-Solomon decoding method**.

The resultant file is **re-transposed** by applying the **decryption function** of the proposed Rubik's Cube transposition cipher used in the upload module.

The decrypted file output is renamed to the selected file name and delivered to the SMF user.

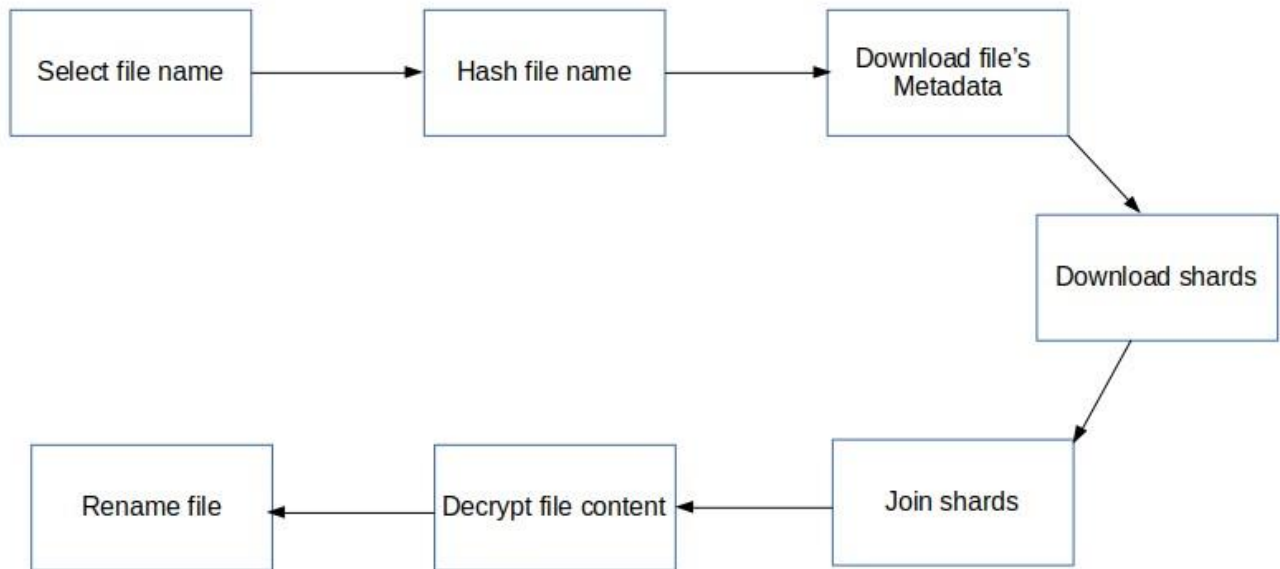


Figure 4.5 - File Download Module

4.13.1. File Downloading Process

When the user selects a file to download, the system downloads the metadata file from SecureMyFiles servers. The system then read the text from the metadata file as a string (Listing 4.44) and then uses a method from the Gson library to create a metadata object that contains all the information in the file (Listing 4.45). The selected file name is hashed by applying the same hash function as that of the upload module to obtain the same hashed value if the file integrity has not been tempered with; the resulting hashed value is used with the file metadata file to obtain the required split chunks or shards from the user's cloud accounts storing them. The downloaded shards are then decoded or joined through using the **Reed-Solomon decoding method**. The resultant file is **re-transposed** by applying the decryption function of the proposed Rubik's Cube transposition cipher used in the upload module. The decrypted file output is renamed to the selected file name and delivered to the SMF user.

```

public static String readFile(String fileName) {
    String fileContent = "";
    try {
        Scanner scan = new Scanner(new java.io.File(fileName));
        while (scan.hasNextLine())
            fileContent += scan.nextLine();
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    }
    return fileContent;
}

```

Listing 4.44 -Method to read text from a file to a string

```
String metadata = FileHandler.readFile("metadata/" + hash +
```

```

".meta");
metadata.File meta = new Gson().fromJson(metadata,
metadata.File.class);
List<Destination> destinations = meta.getDestinations();

```

Listing 4.45 - Snippet of code showing how the system creates a metadata object from a file and accesses the list of destinations from the object

4.14. Reed Solomon Decoding Process

4.14.1. The Error Detection Process

The pseudocode below describes a function to determine whether or not a codeword has been altered (Twum et. al, 2017).

```

::function checkForError(codeword[ ], encodingPolynomial[ ])
::      quotient[ ], remainder[ ]=polynomialDivision(codeword,
encodingPolynomial)
::      if (isZero(remainder))
::          return false
::      else
::          return true
::end function

```

Listing 4.46 - Function to determine data corruption in a codeword

4.14.1.1. Generating the Syndrome Polynomial

A syndrome polynomial is created using values in the finite field which should evaluate to 0 had the codeword retained its integrity. The specific values to use are the same values used to construct the encoding polynomial. The corrupt codeword polynomial is evaluated at the predetermined values and the results from those evaluations are used as the coefficients in the syndrome polynomial. Thus, the degree of the syndrome polynomial is one less than the number of parity values added to the data during encoding. The pseudocode below describes a function to construct the syndrome polynomial for a RS codeword which was encoded using n parity and the values used in creating the encoding polynomial begin with 2^1 (Twum et. al, 2017).

```

::function makeSyndrome(codeword[ ], n) ::      for i=1 to n ::
syndrome[i-1]=evaluate(codeword).at(exponent[i]) ::      end for
::      return syndrome:: end function

```

Listing 4.47 - Function to construct the syndrome polynomial

4.14.1.2. Solving the Key Equation

After the syndrome polynomial is generated, the next step is to determine the error locator polynomial and the error magnitude polynomial for that specific syndrome i.e. for solving the key equation. Various methods have been devised. This study presents a pseudocode for determining the error locator and the error magnitude polynomials based on the Euclid-Sugiyama Algorithm (also known as the Extended Euclidean Algorithm).

```

::function extendedEuclideanAlgorithm(syndrome[ ], n ) ::
initialise rLast[ ] to x^n ::      initialise rCurrent[ ] to
syndrome ::      initialise aLast[ ] and bCurrent[ ] to 1 ::
initialise aCurrent[ ] and bLast[ ] to 0 ::      while (degree of
rCurrent ≥ n/2) ::      q[ ], rNext[
]=polynomialDivision(rLast, rCurrent) ::
aNext=polynomialAddition(aLast,
polynomialMultiplication(aCurrent, q) ::
bNext=polynomialAddition(bLast,
polynomialMultiplication(bCurrent, q)
::      rLast, rCurrent=rCurrent, rNext ::
aLast, aCurrent=aCurrent, aNext ::      bLast,
bCurrent=bCurrent, bNext ::      end while ::
return bCurrent as locatorPolynomial, rCurrent as
magnitudePolynomial
::end function

```

Listing 4.48 - Function for determining error locator and error magnitude polynomials

4.14.1.3. Searching for the Values of the Error Locations

There are two cases when it comes to locating errors using Reed Solomon decoding.

Case 1: Errors (When the locations of the errors are not known before-hand) :- The inverses of the roots of the error locator polynomials are the locations of the errored coefficients in the corrupt codeword. There are a number of algorithms which are able to find the roots of the locator polynomial. One of the most widely used is the Chien Search (REDTITAN, 2011; Cox, 2012).

However, since the possible roots of the polynomial are finite, an extensive search using the elements of the finite field is also a quick and easy-to-program approach to finding the error locations. The pseudocode below describes a function to determine the inverses of the roots of the error locator polynomial.

```

::function findLocations(locatorPolynomial[ ]) ::      for i=0 to
(number of data values+number of parity values-1) ::
result=evaluate(locatorPolynomial).at(1/exponent[i] ) ::
if result=0 ::      locations[ ].include(i) ::
end if ::      end for ::      return locations ::end function

```

Listing 4.49 - Function to determine specific error locations

Case 2: Erasures (When the locations of the errors are known before-hand) :- In the case where the locations of the corruptions are known before-hand, it is unnecessary to perform a search for the error locations. The locations data is simply constituted from the external source that informs of the corruption.

4.14.1.4. Determining the magnitudes of the errors

The last step in generating the error polynomial is to find the magnitudes of the errors at the previously determined locations. The Forney Algorithm (REDTITAN, 2011; Cox, 2012), is an efficient way of doing so. The pseudocode below describes a function to implement the Forney algorithm to find the magnitudes of the errors.

```

::function forneyAlgorithm(locator[ ], magnitude[ ], locations[ ])
::  derivative[ ]=formalDerivativeOf(locator) ::  for i=0 to
length of locations-1 ::      location=locations[i] ::
inverse=1/location ::      error[location] = evaluate(magnitude)
        .at(inverse)/evaluate(derivative)

.at(inverse) ::  end for ::
return error ::end function

```

Listing 4.50 - Function to implement the Forney Algorithm to determine the magnitude of an error

4.14.1.5. Error Correction Procedure using Lookup Table or the Advanced Method

Once the error polynomial has been determined, either using the lookup table or the advanced method, correcting the error in the codeword is done by subtracting the error polynomial from the codeword

polynomial. The result is the corrected codeword which should be perfectly divisible by the encoding polynomial. The pseudocode below describes a function to correct the errors in a corrupt codeword using an error polynomial (Twum et. al, 2017).

```
::function correctError(codeword[ ], errorPolynomial[ ]) ::
return polynomialAddition(codeword, errorPolynomial) ::end
function
```

Listing 4.51 - Function to correct errors in a Reed Solomon codeword

4.15. Checksum Data Recovery Decoding

4.15.1. Reconstruction of the file from file shards

The system read the shards that are downloaded from the cloud and write them into a threedimensional array representing the data. Listing 4.52 shows a code snippet of the function that joins the file shards to form a three-dimensional array.

```
public byte[][][] join(String fileName){      String dir =
"temp/split";      File directory = new File(dir);      if
(!directory.exists())      throw new FileNotFoundException("No
'split' directory");

    byte[] bytes = null;
    int length = FileUtils.readFileToByteArray(new File(dir + "/"
+ fileName + "." + 1)).length;
    byte[][][] data = new
byte[length][4][4];      for (int i = 0; i <
4; i++){      for (int j = 0; j < 4;
j++){
        int a = 4 * i + j;
        bytes = FileUtils.readFileToByteArray(new File(dir +
"/" + fileName + "." + a));
        for (int k = 0; k < bytes.length; k++){
            data[k][i][j] = bytes[k];
        }
    }
}
```

Listing 4.52 – Reading file shards to form a data array

4.15.2. Reading Parity Data from Files

The system read the parity data from three files, one for the row parity, one for the column parity and a last one for the module parity. The data from the files are used to populate a parity object. Listing

4.53 shows a code snippet used for the reading of files into the parity object.

```
public void readFromFiles(String fileName) throws IOException {
    fileName = DIR + "/" + fileName;
    byte[] rowArray = FileUtils.readFileToByteArray(new
    File(fileName + ".row"));
    byte[] colArray = FileUtils.readFileToByteArray(new
    File(fileName + ".col"));
    module = FileUtils.readFileToByteArray(new File(fileName +
    ".module"));
    int length = rowArray.length / 28;
    row = new byte[length][7][4];      column
    = new byte[length][4][7];

    int a = 0;
    for (int i = 0; i < row.length; i++) {          for
    (int j = 0; j < row[i].length; j++) {          for
    (int k = 0; k < row[i][j].length; k++) {
    row[i][j][k] = rowArray[a++];
    }
    }
    }
    a = 0;
    for (int i = 0; i < column.length; i++) {          for
    (int j = 0; j < column[i].length; j++) {          for
    (int k = 0; k < column[i][j].length; k++) {
    column[i][j][k] = colArray[a++];
    }
    }
    }
    del(fileName);
}
```

Listing 4.53 – Reading parity data from files

4.15.3. Error Location

The system re-computes the module parity for the three-dimensional data array and compare the computed parities with the module parity data that was read form the file. The two parity information are compared for equality. Where there is inconsistency, the data is flagged as corrupt and the system proceeds to check the rows and columns for the offending array element (Listing 4.54 and Listing 4.55).

```
void LocateDefectedModule() {
```



```

        boolean errorExist = false;

        for(int i =0; i < data.length; i++){
            if(ComputeModuleSum(data[i]) != moduleParity[i]){
                System.out.println("module "+(i+1)+" has error");
                LocateErrorInModule(this.data[i],i);
                errorExist = true;
            }
        }
        if(!errorExist)
            System.out.println("no error exist");
    }

```

Listing 4.54 – Checking modules for errors

```

        private void LocateErrorInModule(byte[][] d,int x){      int
m,n,p; //declaration of variables          boolean noError = true;
//assigning the error checker a value    for(int row = 0;row <
d.length; row++){ //looping through the rows of the data
            if(!checkSum( d[row] , this.rowsParity[x][row][6] )){
//if the value computed is not true
                noError = false; //assign noError to false to show
that an error exists in the data
                for(int col = 0; col < d[0].length;col++){ //looping
through the columns of the data
                    switch(row){ //checking the row we are working
with
                        case 0: //if its row 1
                            m = d[row][col] ^ d[row+1][col]; //parity
of the value in the row is computed with the value on the next row
                            n = d[row][col] ^ d[row+2][col]; //parity
of the value in the row is computed with the value two rows below it
                            p = d[row][col] ^ d[row+3][col]; //parity
of the value in the row is computed with the value three rows below
it
                            if( m != this.colsParity[x][row][col] )
//checking if the value computed is equal to the value stored in the
colsParity array
                                if( n !=
this.colsParity[x][row+1][col] ) //checking if the value computed is
equal to the value stored in the colsParity array
                                    if( p !=
this.colsParity[x][row+2][col]) { //checking if the value computed
is equal to the value stored in the colsParity array
                                        boolean isNotCorrupt =

```

```
rowCheck(d,row,col,x); //checking if there is a corruption in the  
data
```

```
if(!isNotCorrupt){
```

KNUST



```

d[row][col]
= 0;
//data that is corrupt is set to zero

System.out.println("\tcell
("+row+" , "+col+" )"); //displaying the cell that has an error
    }
    }
    else continue; //moves to the next value if they are the same
    else continue; //moves to the next value if they are the same
    else continue; //moves to the next value if they are the same
    break;

    case 1: //if its row 2
m = d[row][col] ^ d[row-1][col]; //parity
of the value in the row is computed with the value on the row
before it
n = d[row][col] ^ d[row+1][col]; //parity
of the value in the row is computed with the value on the row below
it
p = d[row][col] ^
d[row+2][col]; //parity
of the value in the row is computed with the value two rows below
it
    if( m !=
this.colsParity[x][row-1][col])
//checking if the value computed is equal to the value stored in
the colsParity array
        if(n !=
this.colsParity[x][row+2][col]) //checking if the value computed is
equal to the value stored in the colsParity array
            if( p !=
this.colsParity[x][row+3][col]){ //checking if the value computed
is equal to the value stored in the colsParity array
                boolean
isNotCorrupt =
rowCheck(d,row,col,x); //checking if there is a corruption in the
data

if(!isNotCorrupt){
d[row][col]
= 0;
//data that is corrupt is turned to zero

System.out.println("\tcell
("+row+" , "+col+" )"); //displaying the cell with an error
    }

```



```
    }  
    else continue; //moves to the next value if they are the same  
    else continue; //moves to the next value if they are the same  
    else continue; //moves to the next value if they are the same  
    break;
```

```
case 2: //if it's row 3
```

KNUST



KNUST



```

m = d[row][col] ^ d[row-2][col]; //parity
of the value in the row is computed with the value two rows above
it
n = d[row][col] ^ d[row-1][col]; //parity
of the value in the row is computed with the value one row above it
p = d[row][col] ^
d[row+1][col]; //parity
of the value in the row is computed with the value one row below it
if(m !=
this.colsParity[x][row-1][col])
//checking if the value computed is equal to the value stored in
the colsParity array
if( n !=
this.colsParity[x][row+1][col]) //checking if the value computed is
equal to the value stored in the colsParity array
if( p !=
this.colsParity[x][row+3][col]){ //checking if the value computed
is equal to the value stored in the colsParity array
boolean
isNotCorrupt =
rowCheck(d,row,col,x); //checking if there is a corruption in the
data
if(!isNotCorrupt){
d[row][col] =
0;
//data that is corrupt is turned to zero
System.out.println("\tcell
("+row+" , "+col+" )"); //displaying the cell with an error
}
}
else continue; //moves to the next value if they are the same
else continue; //moves to the next value if they are the same
else continue; //moves to the next value if they are the same
break;

case 3: //if it's row 4
m = d[row][col] ^ d[row-3][col]; //parity
of the value in the row is computed with the value three rows above
it
n = d[row][col] ^ d[row-2][col]; //parity
of the value in the row is computed with the value two rows above it
p = d[row][col] ^
d[row-1][col]; //parity
of the value in the row is computed with the value one row above it

```



```

                                                                    if(m !=
this.colsParity[x][row-2][col])
//checking if the value computed is equal to the value stored in
the colsParity array
                                                                    if(n !=
this.colsParity[x][row+1][col]) //checking if the value computed is
equal to the value stored in the colsParity array
                                                                    if(p !=
this.colsParity[x][row+2][col]){ //checking if the value computed
is equal to the value stored in the colsParity array

```



```

boolean isNotCorrupt =
rowCheck(d,row,col,x); //checking if there is a corruption in the
data

if(!isNotCorrupt){
    d[row][col] = 0;
//data that is corrupt is turned zero
System.out.println("\tcell
("+row+" , "+col+" )");
}

else continue; //moves to the next value if they are the same
else continue; //moves to the next value if they are the same
else continue; //moves to the next value if they are the same
break;
default:
}
}
}

for(int i = d.length-1; i >= 0; i--){
    for(int j = 0; j < d[i].length; j++){
        if(d[i][j] == 0){
            d[i][j] = this.colsParity[x][6][j];
        }
    }
    break;
}

while(countErrors(d) > 0){ //checking if the number of errors
in the module are more than zero
    ResolveCorruption(d, x); //the ResolveCorruption method is
called on the module and its coordinates
}
}

```

Listing 4.55 – Error location within the module

4.15.4. Error Correction

The system resolves the data corruption by looking through the rows parity and columns parity that is read from file and performs an XOR sum at the appropriate locations to correct the error (Listing 4.56).

```

private void ResolveCorruption(byte[][] d, int x){

    for(int row = 0;row < d.length; row++){ //looping through
the data
        for(int col = 0; col < d[row].length;col++){ //looping
through the row of the data

            if( d[row][col] == 0 ){ //checks if the value in the
position we are on in the iteration is zero

                resolve(d,row,col, x); //the resolve method is
called

            }

        }

    }

    for(int row = 0;row<d.length;row++){ //looping through
the data
        if(d[row][d[row].length-1] == 0){ //checking the value in
the last column of the data

            switch(row){ //checking the row

                case 0: //if it's row 1

                    d[row][3] = (byte)((d[row][2] !=
0)?d[row][2]^this.rowsParity[x][row][5]:d[row][3]); //the data in
the last column is checked and if it is zero, the value is computed
with its corresponding value in the rowsParity array and the value
in the column before it else it is maintained

                    d[row][3] = (byte)((d[row+1][3] !=
0)?d[row+1][3]^this.colsParity[x][row][3]:d[row][3]); //the data in
the last column is checked and if it is zero, the value is computed
with its corresponding value in the colsParity array and the value
in the row below it else it is maintained

                    break;

                case 1: //if it's row 2

```



```

d[row][3] =
(byte)((d[row][2] !=
0)?d[row][2]^this.rowsParity[x][row][5]:d[row][3]); //the data in
the last column is checked and if it is zero, the value is computed
with its corresponding value in the rowsParity array and the value
in the column before it

d[row][3] =
(byte)((d[row-1][3] != 0)?d[row-
1][3]^this.colsParity[x][row-1][3]:d[row][3]); //the data in the
last column is checked and if it is zero, the value is computed
with its corresponding value in the colsParity array and the value
in the row above it else it is maintained

d[row][3] =
(byte)((d[row+1][3] !=
0)?d[row+1][3]^this.colsParity[x][3][3]:d[row][3]); //the data in
the last column is checked and if it is zero, the value is computed
with its corresponding value in the colsParity array and the value
in the row below it else it is maintained

break;

case 2: //if it's row 3

d[row][3] =
(byte)((d[row][2] !=
0)?d[row][2]^this.rowsParity[x][row][5]:d[row][3]); //the data in
the last column is checked and if it is zero, the value is computed
with its corresponding value in the rowsParity array and the value
in the column before it

d[row][3] =
(byte)((d[row-1][3] != 0)?d[row-
1][3]^this.colsParity[x][3][3]:d[row][3]); //the data in the last
column is checked and if it is zero, the value is computed with its
corresponding value in the colsParity array and the value in the
row above it else it is maintained

d[row][3] =
(byte)((d[row+1][3] !=
0)?d[row+1][3]^this.colsParity[x][5][3]:d[row][3]); //the data in
the last column is checked and if it is zero, the value is computed
with its corresponding value in the colsParity array and the value
in the row below it else it is maintained

break;

case 3: //if it's row 4

```

```

d[row][3] =
(byte)((d[row][2] !=
0)?d[row][2]^this.rowsParity[x][row][5]:d[row][3]); //the data in
the last column is checked and if it is zero, the value is computed
with its corresponding value in the rowsParity array and the value
in the column before it

d[row][3] =
(byte)((d[row-1][3] != 0)?d[row-
1][3]^this.colsParity[x][5][3]:d[row][3]); //the data in the last

```



column is checked and if it is zero, the value is computed with its corresponding value in the colsParity array and the value in the row above it else it is maintained

```
                break;

        default:
            }
        }
    }
    display(d);
}
```

Listing 4.56 – Error correction

4.15.5. Writing Corrected Data to File

The system finally writes the corrected data to file. Listing 4.57 shows the function that writes the data to file.

```
public void writeToFile() throws IOException
{
    int length = array.length * 16;
    int a = 0;

    byte[] bytes = new
byte[length];
    for (byte[][] b :
array){
        for (byte[] c : b){
            for (byte d : c)
                bytes[a++] = d;
        }
    }
    a = length - 1;
    while (bytes[a] != 1)
        a--;

    FileUtils.writeByteArrayToFile(new File(filename),
Arrays.copyOf(bytes, a + 1));
}
```

Listing 4.57 – writing data from the three-dimensional array to file

4.16. Web Server

Before one can get access to the core functionalities of SecureMyFiles System, a user account needs to be created. The user account details are stored in a database stored on a web server to make the application portable. The SMF System can be used on any computer that is connected to the Internet.

Essential to the SMF System is its metadata which stores pieces of information pertaining to the user, and the uploaded files. The user metadata and the files metadata are also stored on the web server for authenticating users to the application from any computer and also fetch files from the users subscribed cloud accounts.

4.16.1. Database

The SecureMyFiles software makes use of a two-table database which is hosted on a web server. The database is purposefully meant for user authentication to the application. There is a user_account table which has data associated with each account. The other table is the user table which keeps details for each user of the system. There exists a relationship between the tables; the user field in the user_account table is a foreign key to the user_id field in the user table. The database schema is as shown in Figure 4.5.

The log-in details requires a username, which is the user's primary email, and a password. These are bundled together for authentication. If any of the detail is wrong, authentication fails. As an extra layer of security, the user's password is salted before the resulting string is converted into a 256-bit long string using the SHA-256 hashing algorithm.

Java provides a layer of abstraction for relational databases by the use of the Java Persistence Architecture (JPA). JPA is an example of an object-relational (O/R) mapper. This mapper juggles between the entities in the database and the business logic. The O/R mapper is basically a collection of classes called entity classes, with each representing a particular entity in the database. The application thus does not communicate with the database directly but rather, does so through the entity classes. For this to happen, an object of the entity class(es) are created first.

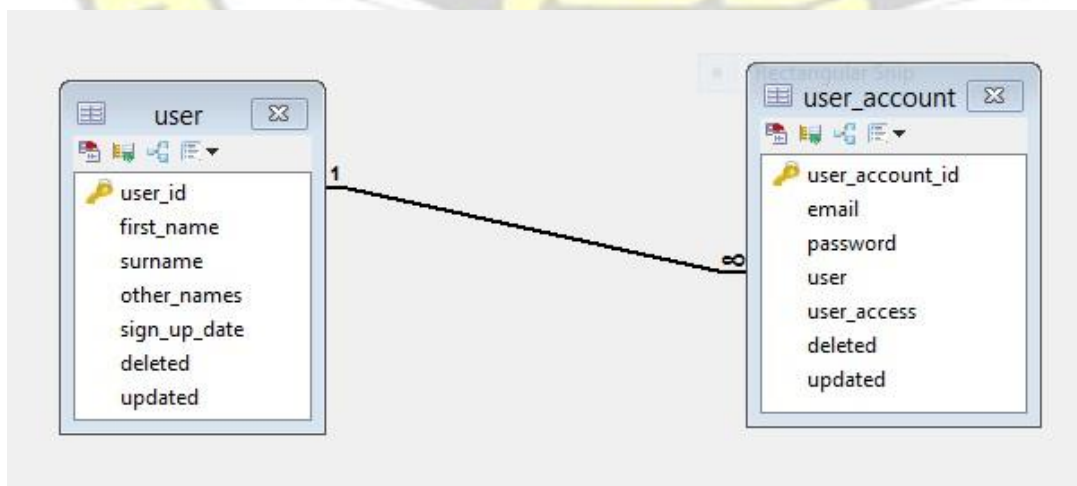


Figure 4.6 - Database schema

A Java class is an entity class when it meets the following criteria:

- The class must be annotated with `@Entity`.
- The class must have a public or protected, no-argument constructor. The class may have other constructors.
- The class, methods, or instance variables must not be declared final.
- If an entity instance is passed by value as a detached object, such as through a session bean's remote business interface, the class must implement the `Serializable` interface.
- Entities may extend entity classes and non-entity classes, and non-entity classes may extend entity classes.
- Persistent instance variable must be declared private, protected, or package-private and can be accessed directly by only the entity class' methods. Clients must access the entity's state through accessor or business methods.

There can be other annotations aside these two. With the annotations, constraints in the actual database can be foregone. For instance, the annotation, `@JoinColumn`, indicates a field corresponding to a field in an entity being a foreign key. With such an annotation in place, even if this constraint is not enforced in the actual database, JPA will ensure the data meets this requirement before it goes into the database.

Actual database transaction occurs by means of the `EntityManager` API. With the entity manager, an entity object, which corresponds to a row in a database table, can be created, updated, removed, and deleted. An object-oriented query language called Java Persistence Query Language (JPQL), which is similar to SQL, is issued by the entity manager. The entity manager however, converts the JPQL queries to SQL queries so that the JPA can actually interact with the database.

The Java Persistence consists of the: Java Persistence API, Java Persistence Query Language, O/R mapping metadata.

CHAPTER 5

TESTING, RESULTS AND DISCUSSIONS

This chapter presents the results from the testing of the various sub-systems that were implemented and integrated to create the SMF system. The system was passed through various testing stages including component testing, integration testing, and system testing. The beta version of the SMF system has been deployed on various end-user client nodes for the acceptance testing. The chapter also presents a discussion of the results and how it addresses the study objectives.

5.0. Testing and Results

The sections that follow disclose the process used to test the proposed system and the results obtained from the tests. The testing was carried out in an experimental lab set-up using JAVA, SQL, and PHP software development tools installed on a very high-spec PC (64-bit Intel Core-i7 CPU running at 3.60GHz, 12.0GB RAM) and Laptop (64-bit Intel Core-i7 CPU running at 2.20GHz, 8.0GB RAM). In addition the experimental setup required a stable computer network infrastructure.

Various testing scenarios were set-up for experimentations to evaluate the SMF system capability of recovering a file in an event of corruption (whether shards modifications or shards deletion). Results from the experiments conducted using the Reed Solomon Erasure protection and also this studies newly developed Checksum Data Recovery (CDR) Erasure protection under different testing scenarios are presented later in this chapter.

5.1. Cloud Providers

There are a number of cloud service providers that provide premium and free services. Those that are available for use with the SMF App are Dropbox, Google Drive, and Box. Each of these cloud accounts is connected to the SMF App via their respective Java APIs. These APIs provide safe and secure connections to the cloud accounts just as it is when accessing an account on the cloud provider's website.

A user needs to sign up to a minimum of two cloud accounts—Dropbox, Google Drive, and Box—before using the SMF App. For testing the proposed system, three cloud accounts were used. Due to security, and file recovery constraints, the user should have three cloud accounts registered with the application in the worse-case scenario. However, six is highly recommended for a good balance between performance and security.

5.2. File Uploading Sequence

1. File Selection
2. File Name Obfuscation
3. Data Encryption
4. Erasure Protection
5. Data Dispersal
6. File Upload

5.2.1. File selection

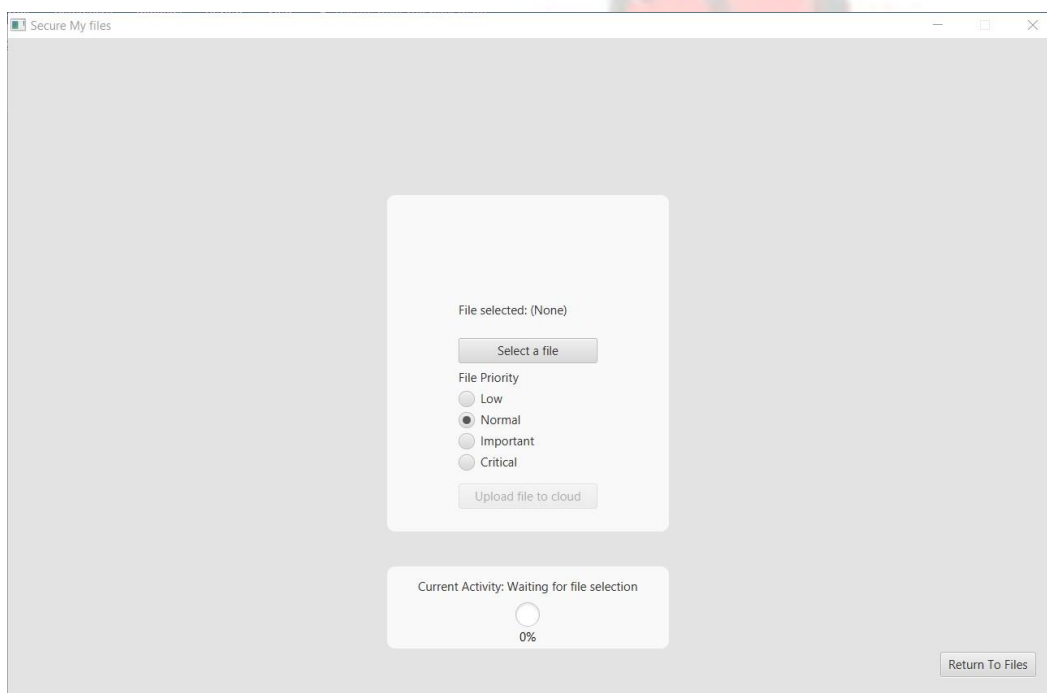


Figure 5.1 - User Interface for selecting and configuring file for upload

The system uses Java's *JavaFX FileChooser UI* to present the user with an interface with which a file can be selected to be used.

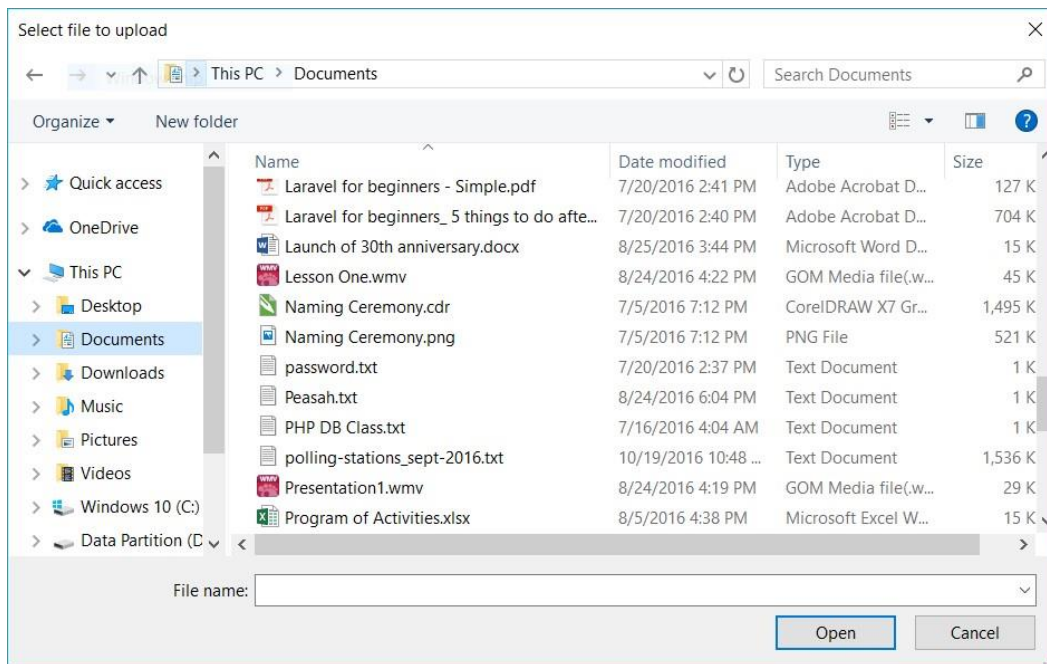


Figure 5.2 - User Interface for choosing a file for upload

5.2.1.1. Constraints

File type constraints: The selection module accepts all file types.

File size constraints: The selection module does not enforce any file size constraints. It should be noted though that the Java Virtual Machine (JVM) allocates memory for the program based on the hardware Random Access Memory (RAM). The implication here is that files with a substantially large size can cause an *OutOfMemory Exception* during runtime.

5.2.2. File name obfuscation

The first line of security that the system provides is a hashing of the file name. All subsequent operations on the file are done with the hashed name instead of the original file name. This helps to hide the purpose of the file from any intruder who manages to break into the user's cloud account and access the file's pieces online.

The system uses Secure Hash Algorithm 1 (SHA1) from Java's inbuilt Security package. Though the Message Digest 5 (MD5) was considered as an alternative hashing method but the MD5 algorithm is now widely regarded as cryptographically broken (Kessler, 2017). This means that the hash from an MD5 function can be reversed to obtain the original data which was hashed. As such, the MD5 algorithm was discarded in favour of SHA1 which is still a one-way hashing scheme. Further, SHA1 produces a longer string than MD5. This is useful in that it is more difficult to break a longer string than a shorter one.

The result from implementation of the hashing activity is shown in figure 5.3.

```
"C:\Program Files\Java\jdk1.8.0_121\bin\java" ...  
Original file name: '256-byte file.txt'  
Hashed file name: '55e7850d19339db27bf8262bd819856afdc20de8'  
  
Process finished with exit code 0
```

Figure 5.3 - Result from hashing file name.

5.2.3. Data encryption

The second line of security that the system provides is an encryption module to obfuscate the data within the file. The system makes use of a proposed transposition cipher that mimics the motions of the Rubik's Cube. The system writes the file's byte data onto the face of a virtual customized Rubik's Cube and uses a custom algorithm described in Chapter 3 Section 3.2.1.2 to create a sequence of rotations to obfuscate the data.

5.2.3.1. Encryption Key

The system uses a hash value based on the user's credentials as the encryption key. The system applies a custom algorithm to transform the hash value into a sequence of rotations.

5.2.3.2. Encryption

During an encryption, the rotation sequence is followed forwards and each rotation is carried out in the clockwise direction.

5.2.3.3. Decryption

During a decryption, the rotation sequence is read from the last to the first and each rotation is carried out in the counter-clockwise direction. This undoes the clockwise rotations done during the encryption.

Following the process outlined in Section 4.5, the content of the file was encrypted using this study's proposed transposition cipher which is the Rubik's Cube algorithm and the results obtained are shown in figure 5.4.


```

C:\Program Files\Java\jdk1.8.0_121\bin\java" ...
Content before encryption:
Mary had a little lamb. Little lamb, little lamb. Mary had a little lamb. Its fleece was white as snow
And everywhere that Mary went. Mary went, Mary went. Everywhere that Mary went. The lamb was sure to go

Rubik's Cube File length: 207

Content after encryption:
Mmyt d t littme l t
weitrlxrlar,.ain leblamb,aMaayhhdasa little la b eLrw flaec wes whnre as tnoa
Atd tvhmysere Mha Merytwen . Maay ent,eMaTy we . Evetyw era tl .lwhy ren . thewtitrLtaslsure bo goma w

```

Figure 5.4 - Result from encrypting file content

5.2.4. File splitting with erasure protection

Every file is split into a predetermined number of shards before upload. Reed Solomon Coding and the Checksum Data Recovery were used to enforce erasure protection. To make use of Reed Solomon Coding, first an encoding polynomial was generated after which it was used to encode the data in a file. Figure 5.5 shows a list of encoding polynomial coefficients from parity size 1 to parity size 32. Figure 5.6 shows the encoding polynomial coefficients for parity size 32 which may be useful for data shards up to 223. Table 5.1 shows the results from encoding a 256-byte file into 32 data shards and 8 parity shards.

```

Run - ReedSolomon
C:\Program Files\Java\jdk1.8.0_121\bin\java" ...
1: (2, 1)
2: (5, 6, 1)
3: (44, 84, 24, 1)
4: (116, 251, 216, 30, 1)
5: (38, 197, 229, 63, 62, 1)
6: (117, 48, 38, 159, 4, 126, 1)
7: (24, 208, 125, 146, 164, 245, 254, 1)
8: (37, 244, 4, 172, 21, 229, 44, 227, 1)
9: (159, 92, 49, 114, 44, 235, 132, 217, 1)
10: (160, 212, 68, 182, 30, 197, 190, 140, 47, 173, 1)
11: (97, 180, 209, 153, 135, 186, 219, 7, 132, 50, 69, 1)
12: (120, 252, 175, 132, 170, 147, 147, 130, 61, 34, 193, 136, 1)
13: (163, 73, 155, 61, 46, 129, 31, 52, 163, 15, 70, 57, 15, 1)
14: (124, 134, 159, 43, 175, 34, 77, 34, 46, 180, 25, 183, 216, 28, 1)
15: (59, 31, 45, 79, 170, 131, 184, 97, 109, 119, 166, 226, 95, 85, 58, 1)
16: (79, 44, 81, 200, 49, 182, 56, 27, 222, 187, 126, 104, 21, 102, 82, 118, 1)
17: (146, 75, 9, 40, 89, 206, 156, 213, 9, 249, 46, 233, 70, 211, 162, 21, 238, 1)
18: (179, 31, 82, 180, 25, 114, 230, 99, 141, 254, 171, 51, 136, 97, 47, 209, 203, 195, 1)
19: (174, 11, 114, 11, 205, 41, 63, 132, 340, 229, 115, 3, 229, 217, 186, 213, 209, 32, 153, 1)
20: (89, 166, 244, 122, 150, 179, 71, 217, 139, 247, 174, 178, 100, 39, 229, 97, 80, 211, 222, 45, 1)
21: (145, 100, 124, 182, 46, 89, 233, 170, 209, 209, 69, 89, 113, 62, 127, 154, 210, 214, 184, 235, 88, 1)
22: (71, 110, 51, 75, 75, 203, 217, 31, 110, 117, 177, 178, 161, 244, 237, 117, 209, 67, 207, 108, 246, 178, 1)
23: (117, 229, 60, 67, 180, 89, 88, 150, 189, 129, 200, 74, 3, 230, 112, 3, 101, 189, 80, 226, 164, 13, 123, 1)
24: (183, 104, 189, 208, 173, 79, 49, 133, 251, 123, 46, 147, 188, 150, 174, 232, 238, 4, 197, 195, 20, 33, 197, 244, 1)
25: (94, 117, 56, 170, 58, 124, 56, 191, 149, 124, 9, 216, 240, 97, 121, 183, 143, 194, 90, 157, 255, 119, 115, 136, 247, 1)
26: (117, 125, 239, 227, 44, 47, 236, 157, 235, 128, 74, 207, 194, 171, 106, 236, 178, 87, 3, 51, 163, 208, 64, 209, 204, 241, 1)
27: (197, 255, 181, 3, 150, 239, 139, 62, 45, 149, 223, 170, 146, 101, 105, 206, 231, 131, 45, 14, 230, 152, 247, 229, 232, 244, 253, 1)
28: (170, 37, 161, 208, 84, 235, 155, 161, 104, 58, 111, 148, 159, 131, 212, 43, 115, 83, 28, 147, 165, 124, 44, 122, 208, 224, 96, 229, 1)
29: (180, 20, 126, 117, 182, 110, 36, 182, 239, 234, 236, 173, 93, 239, 221, 97, 142, 155, 41, 78, 8, 12, 111, 96, 228, 98, 101, 95, 213, 1)
30: (89, 69, 153, 116, 176, 117, 111, 75, 73, 233, 242, 233, 65, 210, 21, 139, 103, 173, 67, 119, 105, 210, 174, 110, 74, 69, 228, 82, 255, 181, 1)
31: (88, 244, 195, 77, 89, 164, 46, 54, 143, 60, 19, 135, 99, 158, 169, 146, 158, 127, 186, 10, 151, 182, 153, 53, 247, 231, 153, 175, 1, 53, 117, 1)
32: (45, 216, 239, 24, 253, 194, 21, 40, 197, 96, 163, 210, 227, 134, 224, 139, 113, 104, 1, 209, 164, 82, 43, 15, 232, 246, 142, 50, 189, 29, 232, 1)

Process finished with exit code 0

```

Figure 5.5 - Encoding Polynomial Coefficients for Parity sizes from 1 up to 32

```

4294967291
4294967292
4294967293
4294967294
4294967295
232 29 189 50 142 246 232 15 43 82 164 238 1 158 13 119 158 224 134 227 210 163 50 107 40 27 104 253 24 239 216 45
Process finished with exit code 0

```

Figure 5.6 - Encoding Polynomial Coefficients for Parity size of 32

Table 5.1 - Encoding of 256-byte file with 32 data shards and 8 parity shards

140	13	247	189	11	50	88	9
18	72	99	140	82	27	12	64
191	156	249	137	226	253	200	156
98	254	122	97	215	28	84	128
116	94	59	74	126	60	135	138
104	247	86	47	73	158	116	254
133	98	34	54	52	178	161	32
49	144	243	199	171	20	36	74
78	111	116	101	112	97	100	43
43	32	73	110	115	116	97	108
108	101	114	32	51	50	45	98
105	116	32	120	56	54	58	32
84	97	107	101	32	116	104	105
115	32	111	110	101	32	105	102
32	121	111	117	32	104	97	118
101	32	110	111	32	105	100	101
97	32	119	104	105	99	104	32
111	110	101	32	121	111	117	32
115	104	111	117	108	100	32	116
97	107	101	46	13	10	78	111
116	101	112	97	100	43	43	32
122	105	112	32	112	97	99	107

Original File Data (Data Shards)

97	103	101	32	51	50	45	98	
105	116	32	120	56	54	58	32	
68	111	110	39	116	32	119	97	
110	116	32	116	111	32	117	115	
101	32	105	110	115	116	97	108	
108	101	114	63	32	67	104	101	
99	107	32	116	104	105	115	32	
111	110	101	32	40	122	105	112	
32	102	111	114	109	97	116	41	
46	13	10	78	111	116	101	112	
97	100	43	43	32	55	122	32	Parity Data
112	97	99	107	97	103	101	32	
51	50	45	98	105	116	32	120	
56	54	58	32	68	111	110	39	
116	32	119	97	110	116	32	116	
111	32	117	115	101	32	105	110	
115	116	97	108	108	101	114	63	
32	55	122	32	102	111	114	109	

After the erasure protection techniques have been applied, the shards are saved as individual files on the disk. Figure 5.7 shows the results of splitting one file.

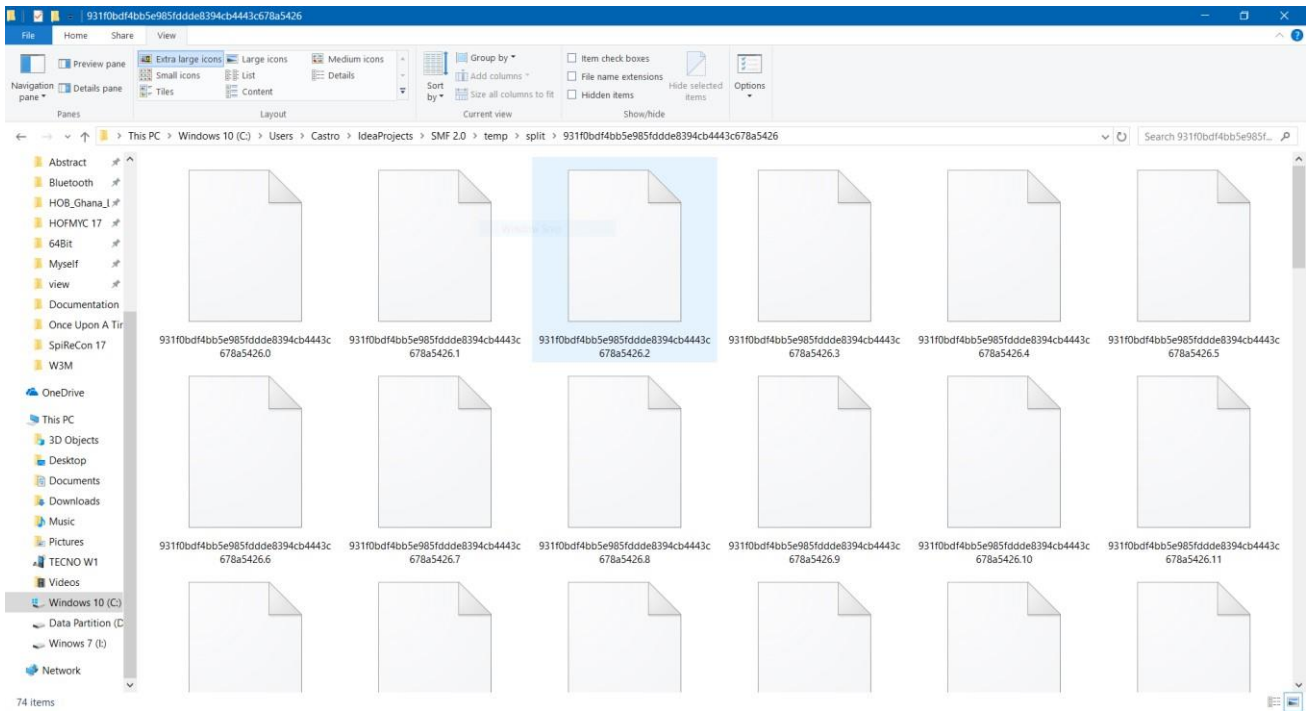


Figure 5.7 - A view into a folder showing the shards of a file after splitting

5.2.5. Data dispersal

The third security measure employed by the system is to scramble the file shards before uploading them to the cloud. The system uses a shuffle function found in Java's Collections package. The function reorders the file shards and assigns the scrambled shards to the user's cloud accounts.

Since the data is uploaded in no particular order, it is difficult to determine the total number of shards from the content of one drive. That total number of shards is very necessary if an intruder wants to reconstruct the files from its shards.

The shuffling method implemented by following the processes outlined in Chapter 4 Section 4.10 gave the results in figure 5.8.

```
interest
"C:\Program Files\Java\jdk1.8.0_121\bin\java" ...
Filenames Before shuffling:
965d3531-fa7b-4a7c-a17f-496a81586df8-1
965d3531-fa7b-4a7c-a17f-496a81586df8-2
965d3531-fa7b-4a7c-a17f-496a81586df8-3
965d3531-fa7b-4a7c-a17f-496a81586df8-4
965d3531-fa7b-4a7c-a17f-496a81586df8-5
965d3531-fa7b-4a7c-a17f-496a81586df8-6
965d3531-fa7b-4a7c-a17f-496a81586df8-7
965d3531-fa7b-4a7c-a17f-496a81586df8-8
965d3531-fa7b-4a7c-a17f-496a81586df8-9
965d3531-fa7b-4a7c-a17f-496a81586df8-10

Filenames After shuffling:
965d3531-fa7b-4a7c-a17f-496a81586df8-5
965d3531-fa7b-4a7c-a17f-496a81586df8-6
965d3531-fa7b-4a7c-a17f-496a81586df8-8
965d3531-fa7b-4a7c-a17f-496a81586df8-4
965d3531-fa7b-4a7c-a17f-496a81586df8-10
965d3531-fa7b-4a7c-a17f-496a81586df8-9
965d3531-fa7b-4a7c-a17f-496a81586df8-1
965d3531-fa7b-4a7c-a17f-496a81586df8-3
965d3531-fa7b-4a7c-a17f-496a81586df8-2
965d3531-fa7b-4a7c-a17f-496a81586df8-7

Process finished with exit code 0
```

Figure 5.8 - Result from file name shuffling

5.2.6. File upload

The final phase of the upload process is to connect to the cloud service through the service provider's API and send the file over secure HTTP into the user's cloud account.

Depending on the SMF user choice of a file priority level (Low, Normal, important, or critical) as depicted by figure 5.1 of Section 5.2.1, the system distributes shards to the user's cloud accounts in accordance to the data and parity computation as set out in Chapter 4 Section 4.8.1.

Testing and results from the selection of the '*Normal*' and '*Important*' file priority levels which employs the Reed Solomon Erasure protection for file recovery under different scenarios are presented (See Scenarios 1 and 2).

The test results from the *critical* priority level which uses this study's proposed Checksum Data Recovery (CDR) described in Chapter 3 Section 3.3.2 and implemented in Chapter 4 Section 4.9 is also presented (See Scenario 3). By choosing the critical file priority option, the SMF user may lose

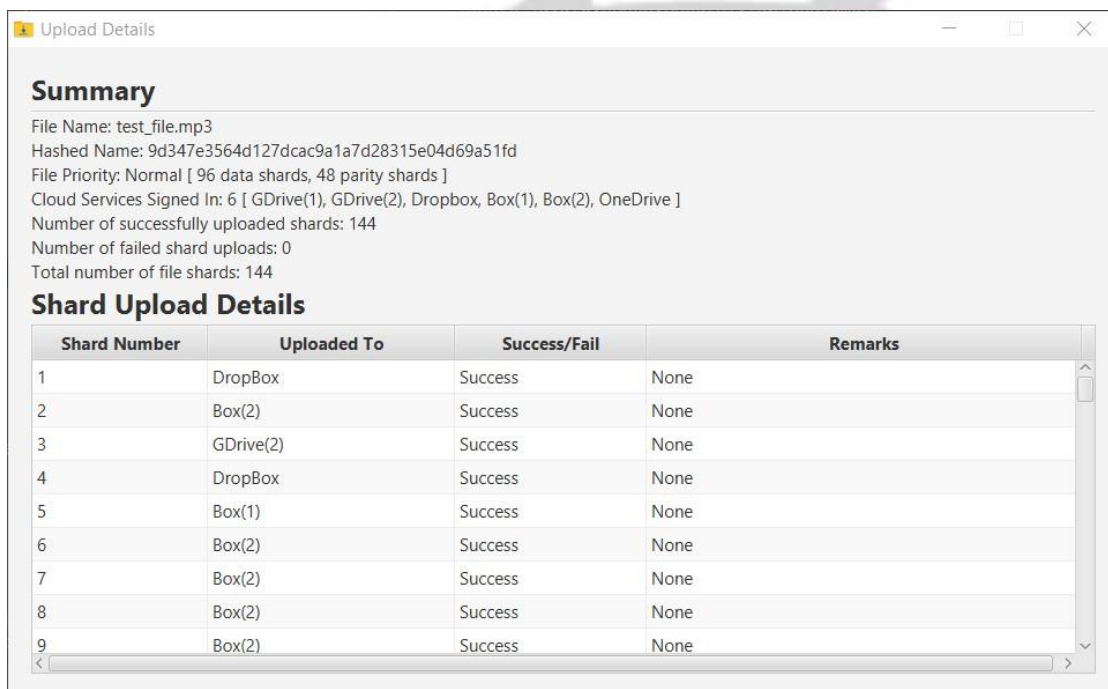
up to 12 shards out of the 16 total shards for any file size uploaded (**Refer to figure 4.4 in Chapter 4, Section 4.9.2**) and still be able to recover the corrupted file.

Scenario 1: Uploading a file by choosing the ‘Normal’ file priority level.

The ‘Normal’ priority level splits a file into 96 data shards and 48 parity shards that are used for file recovery in the event of corruption. This result with a total of 144 shards that are uploaded using the data dispersal method to the six CSP’s each receiving 24 shards.

Case 1

Figure 5.9 depicts a successful file upload operation where the SMF user selected the ‘Normal’ priority level. The figure indicates none of the distributed shards are corrupted.



Shard Number	Uploaded To	Success/Fail	Remarks
1	DropBox	Success	None
2	Box(2)	Success	None
3	GDrive(2)	Success	None
4	DropBox	Success	None
5	Box(1)	Success	None
6	Box(2)	Success	None
7	Box(2)	Success	None
8	Box(2)	Success	None
9	Box(2)	Success	None

Figure 5.9 – A successful file upload choosing the Normal file priority level

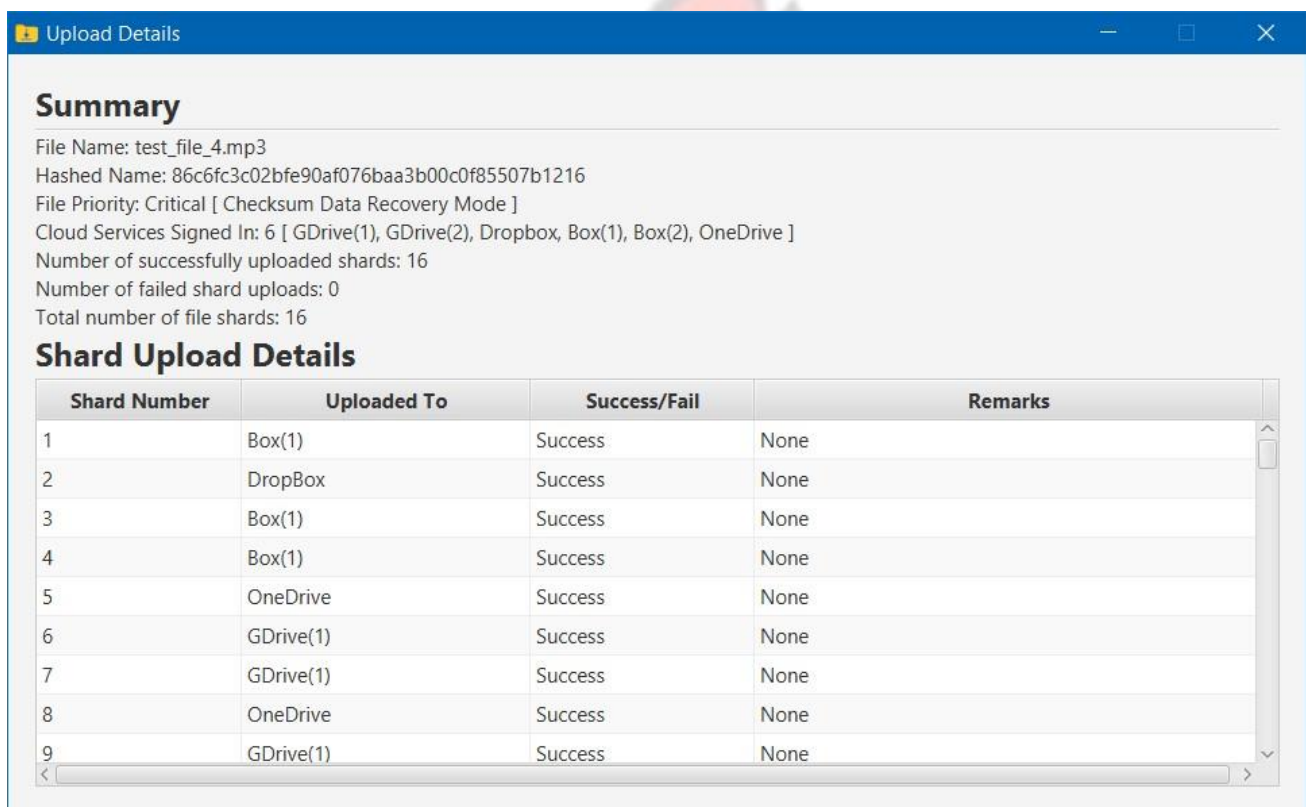
Scenario 2: Uploading a file by choosing the ‘Critical’ file priority level.

As noted in Chapter 2 Section 2.8, Reed Solomon coding can recover any number of data errors up to half the number of parity data stored and can correct any number of erasures up to the number of parity data stored. This feature of Reed Solomon coding places some limitation on the number of data that

can be successfully recovered in the event of data corruption. The proposed Checksum Data Recovery (CDR) which although cannot recover data in the event of total deletion without relying on backup can in most cases recover a file to some extent if at least four of the data shards exist. The ‘**Critical**’ file priority option uses the CDR for erasure protection. Case 1 depicts a successful file uploading operation where the SMF user selected the ‘**Critical**’ priority level.

Case 1

Figure 5.10 depicts a successful file upload operation where the SMF user selected the ‘**Critical**’ priority level. The figure indicates none of the distributed shards are corrupted.



Summary			
File Name: test_file_4.mp3			
Hashed Name: 86c6fc3c02bfe90af076baa3b00c0f85507b1216			
File Priority: Critical [Checksum Data Recovery Mode]			
Cloud Services Signed In: 6 [GDrive(1), GDrive(2), Dropbox, Box(1), Box(2), OneDrive]			
Number of successfully uploaded shards: 16			
Number of failed shard uploads: 0			
Total number of file shards: 16			
Shard Upload Details			
Shard Number	Uploaded To	Success/Fail	Remarks
1	Box(1)	Success	None
2	DropBox	Success	None
3	Box(1)	Success	None
4	Box(1)	Success	None
5	OneDrive	Success	None
6	GDrive(1)	Success	None
7	GDrive(1)	Success	None
8	OneDrive	Success	None
9	GDrive(1)	Success	None

Figure 5.10 – A successful file upload choosing the Critical file priority level

The results from the implementation of the data dispersal method described in Chapter 4, Section 4.10 are shown by figure 5.11 and figure 5.12.

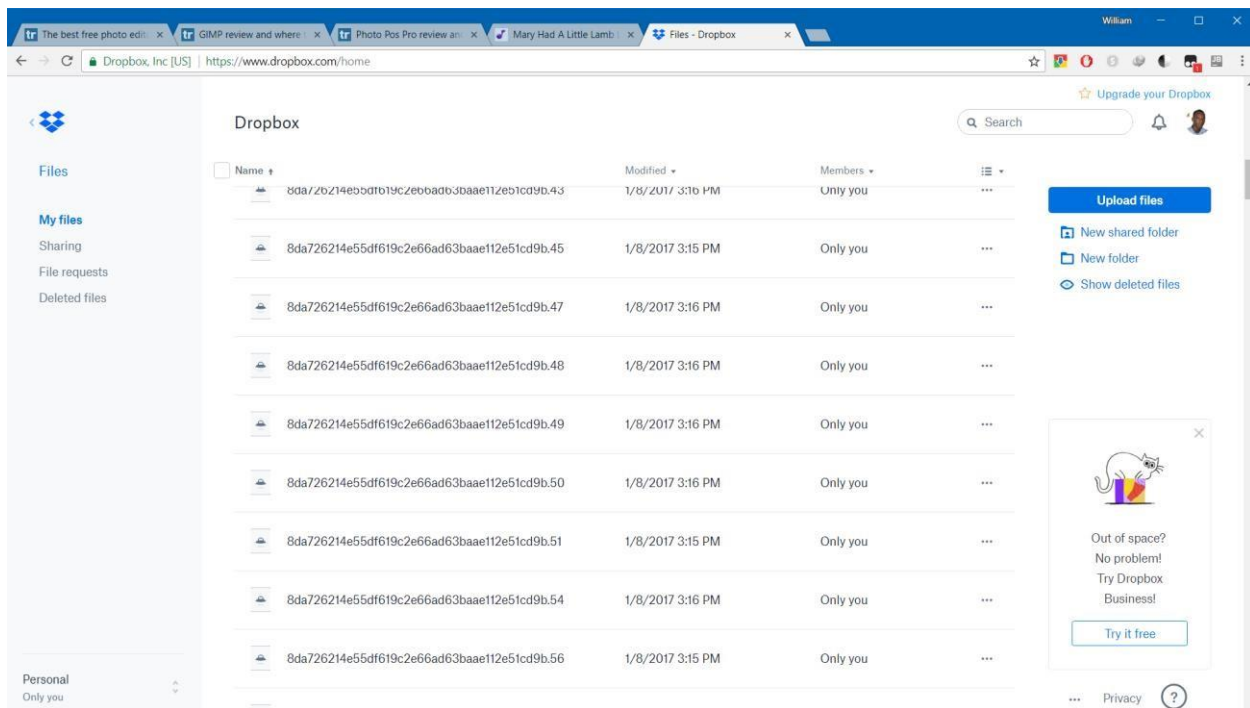


Figure 5.11 - Content of Dropbox account showing some of the shards from a file whose name has been obfuscated.

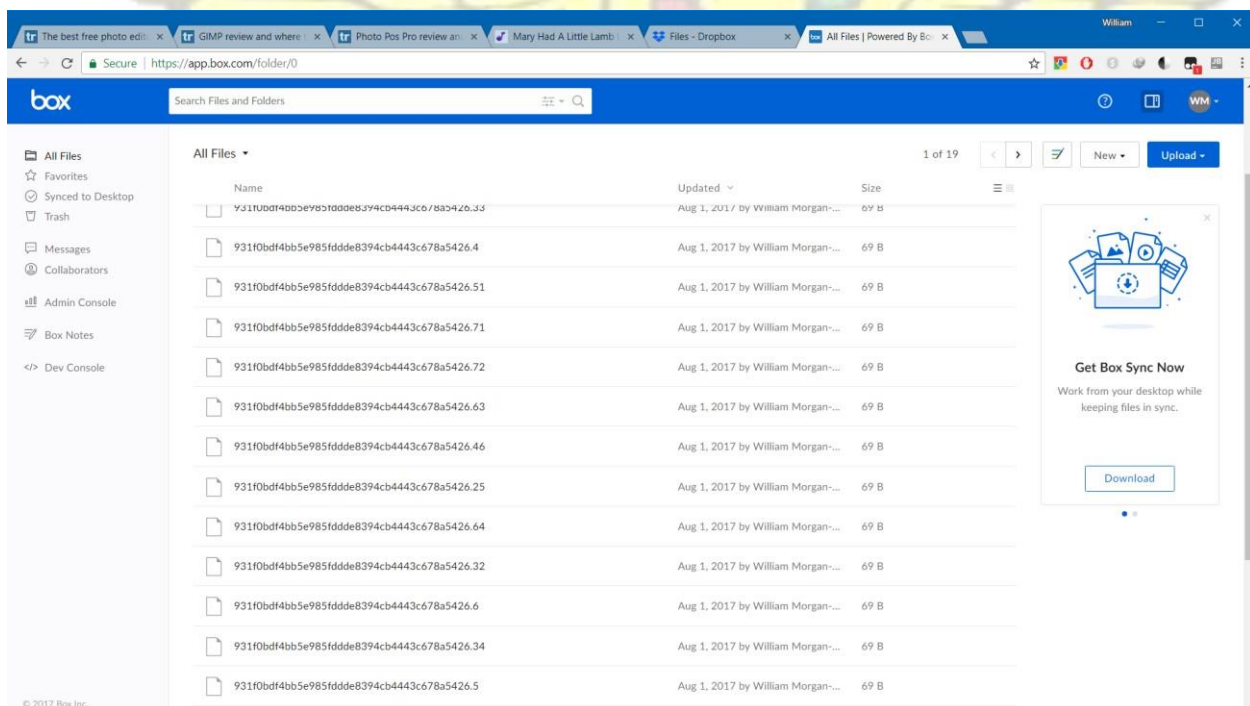


Figure 5.12 - Content of Box account showing some of the shards from a file whose name has been obfuscated.

5.3. File Downloading Sequence

The file downloading sequence follows the process outlined and implemented by the file download module presented in Chapter 4, Section 4.13 as follows:

1. File Selection
2. Data Gathering
3. Erasure Protection
4. Data Decryption
5. File Name Restoration

5.3.1. File Selection

The system displays a list of file names for the files which have been uploaded to the cloud by the user. From the list, the user can select a file for download by clicking on it. After selecting the file, the user then clicks a download icon from the interface icon bar to initiate the download.

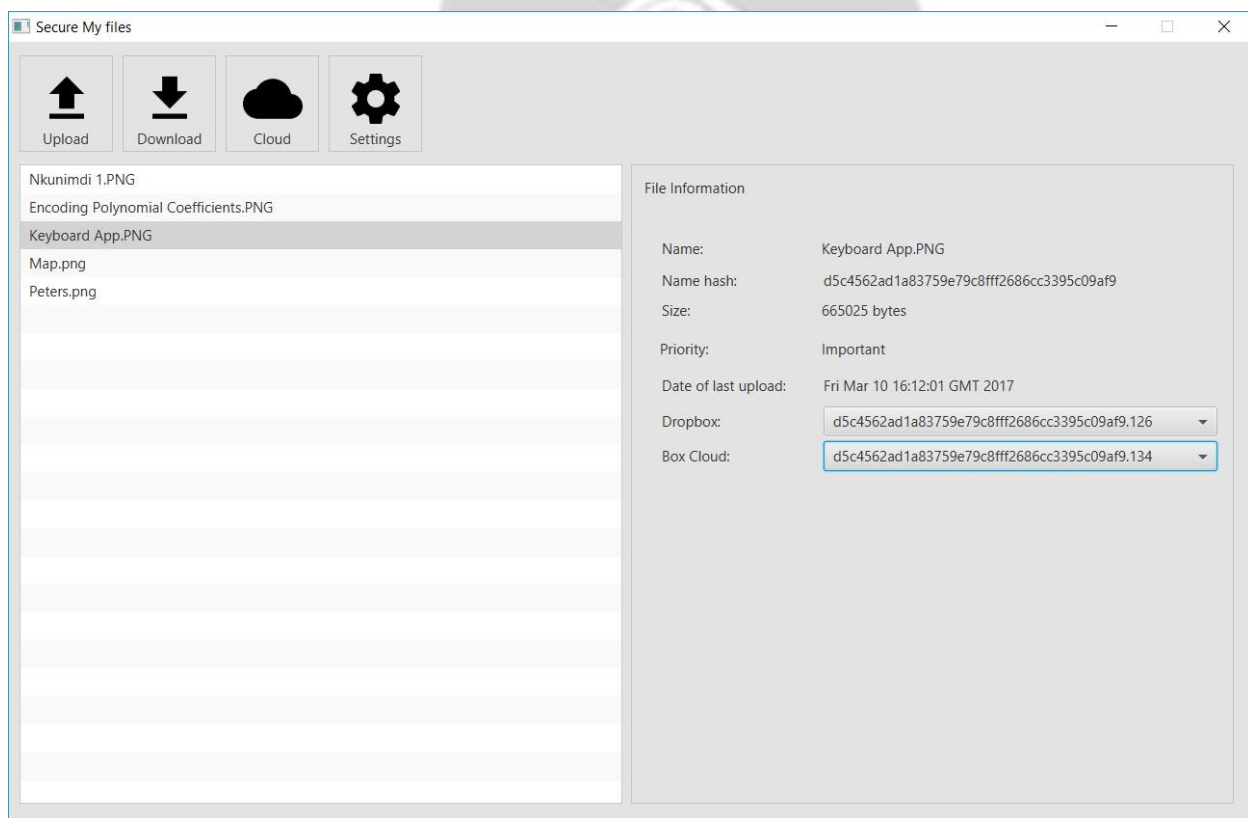


Figure 5.13 - Interface of SMF application showing a list of uploaded files

5.3.2. Data gathering

After the file download has been initiated, the system produces the SHA1 hash of the selected file name and consults the file's metadata to fetch a list of destinations to which the file's shards were dispersed during the file upload. The system then connects to the cloud and downloads the file shards from their respective destinations into a temporary directory (**buffer area**). The downloaded shards include both the original file data as well as the computed parity data. The cloud download requests use the hash value of the file's name instead of the original file name since the file shards were renamed to the hash value before they were uploaded.

5.3.3. Erasure recovery

The system proceeds to verify the integrity of the downloaded file by making use of the erasure protection method that the user selected when uploading the file. The file priorities "low", "normal" and "important" make use of Reed Solomon coding whereas the "critical" file priority makes use of the Checksum Data Recovery (discussed in section 4.6).

5.3.3.1. Reed Solomon Decoding

The system reads the file shards into a two-dimensional array with each row in the array containing the data from one file shard. The data shards are read first into the array before the parity data. The system then iterates through the columns of the two-dimensional array and extracts each column data into a polynomial. The system then takes the remainder when the polynomial is divided by the encoding polynomial. A remainder of zero indicates that the column data is not corrupt. Any other remainder indicates a corruption of the column data. In the case where the data is not corrupt, the system simply proceeds to the next column to test the integrity of that column; otherwise the system takes measures to correct the corruption. The Reed Solomon Error Correction process that the system uses comprises the following steps:

5.3.3.2. Syndrome Polynomial

The system creates a syndrome polynomial by evaluating the column polynomial at the first "parity" values of the Galois Field. Since the syndrome is computed with an erroneous column data, the syndrome is able to quantify the error in the column polynomial.

5.3.3.3. Extended Euclidean Algorithm

The system passes the syndrome polynomial into a method which performs the Extended Euclidean Algorithm using the syndrome as its initial value. The method solves the Reed Solomon key equation

using the Sugiyama-Euclidean Algorithm and return two objects, representing the error locator and error magnitude polynomials.

5.3.3.4. Extensive Search

After having computed the error locator polynomial, the system evaluates the polynomial at the inverses of all the values of Galois Field. The system locates the positions of the errors as the exponents whose values' inverses evaluate to zero.

5.3.3.5. Forney Algorithm

The system then computes the magnitude of error at each of the locations found in the extensive search. The system passes the error locator polynomial, error magnitude polynomial and the locations of the errors as arguments to a method which implements the Forney algorithm to generate an error polynomial representing the alterations to the original file and parity column.

5.3.3.6. Error Correction

The system corrects the alteration in the file and parity column by performing a polynomial addition of the error polynomial and the column polynomial. The resulting polynomial is the original polynomial before the file shards were uploaded. The data from the corrected column is used to replace the data from the corrupt column in the two-dimensional array that holds the file data.

5.3.3.7. File Reconstruction

After the downloaded shards have been verified and corrected, the system reads the byte data from the first "data" rows of the two-dimensional array into a single byte array which is subsequently written to a temporary file.

5.3.4. Data decryption

The system writes the byte data from the temporary file onto the face of a virtual customized Rubik's Cube. The system recreates the sequence of rotations that was used to encrypt the original file, then iterates through the sequence from the last rotation to the first, performing each rotation in the counter-clockwise direction. The process restores the file's data to the right order, making it readable once again.

5.3.5. File name restoration

The system finally moves the temporary file into the system's download directory and renames it to the original file's name.

Scenario 1: Downloading a file uploaded using the 'Normal' file priority level.

Case 1

Figure 5.14 depicts file reconstruction during download for a scenario where 24 of the distributed shards are corrupted or an event where one of the six CSP's refuses to grant a subscriber access to shards stored on their storage servers in the event of a dispute over say subscription payments or for any other reason.

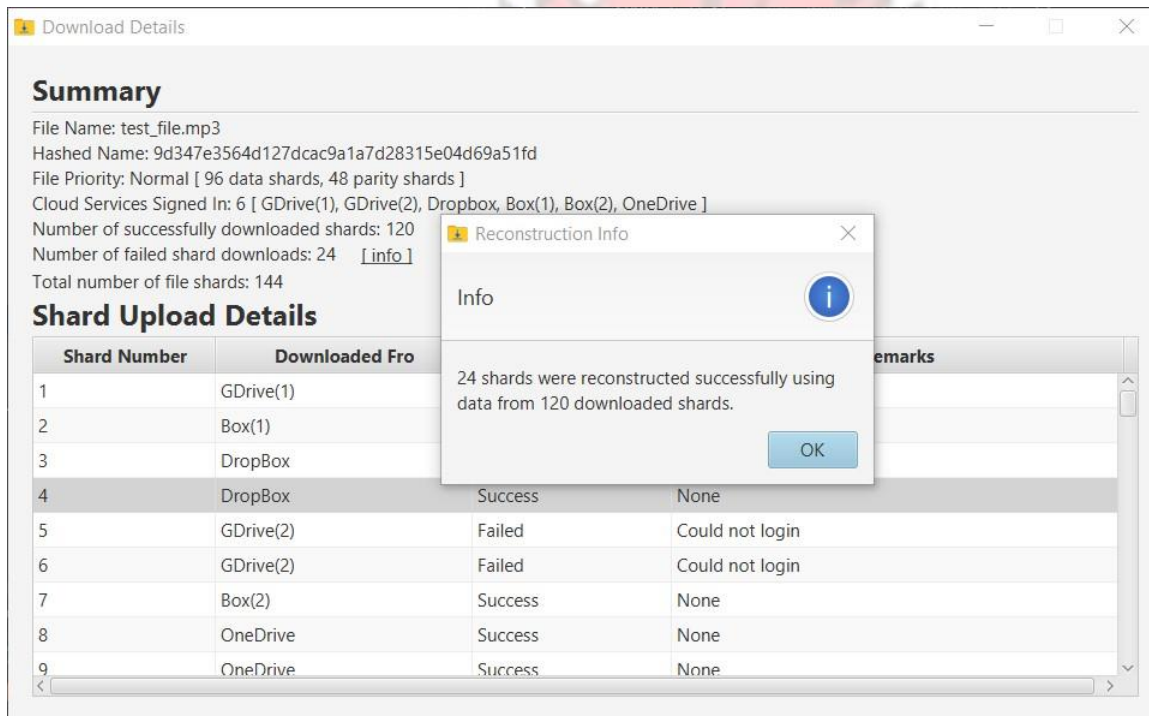


Figure 5.14 – successful file reconstruction during download for 24 corrupted shards with Normal option

Case 2

Figure 5.15 depicts a situation where 48 shards are corrupted or where two of the CSP's systems are down but the SMF system is able to recover the full file and present to the owner.

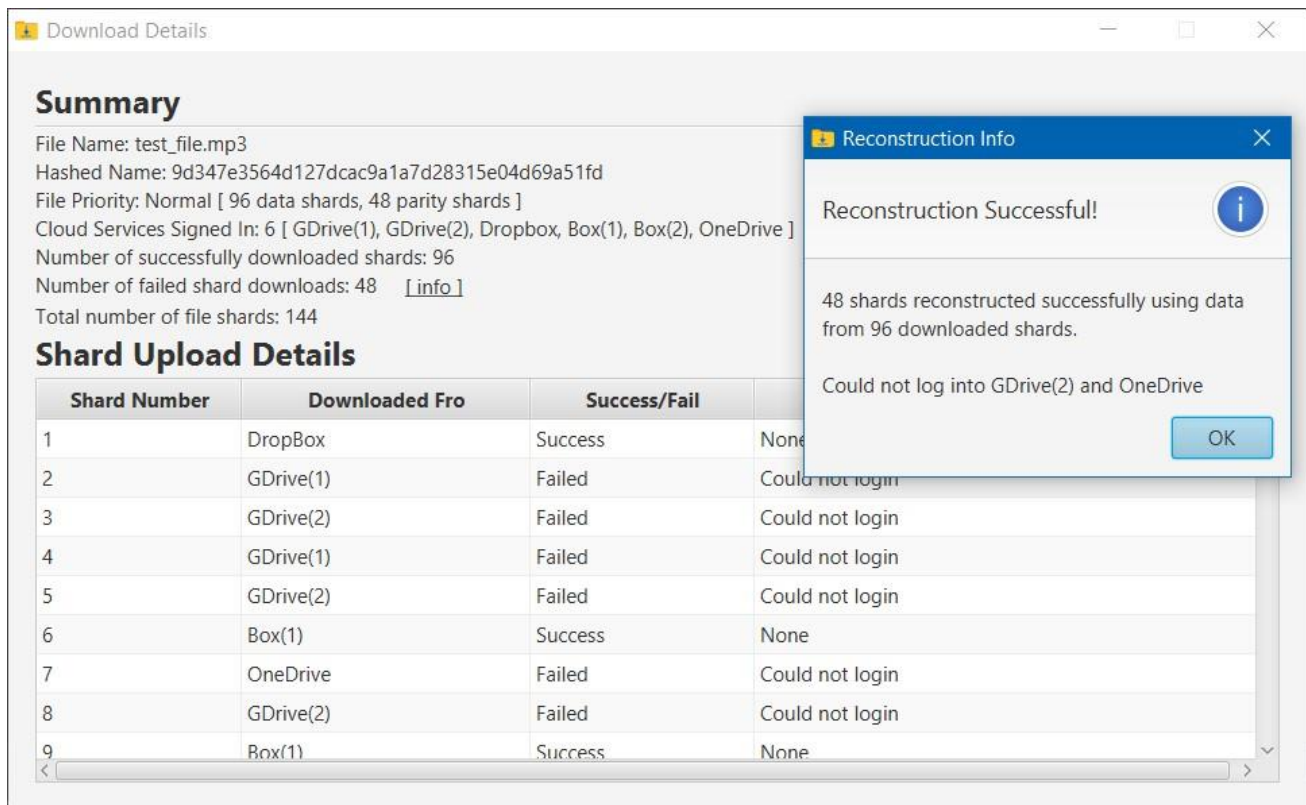


Figure 5.15 – successful file reconstruction during download for 48 corrupted shards with Normal option

Case 3

Unlike Case 1 and Case 2, Case 3 presents a scenario that depicts a situation where more than 48 shards are corrupted i.e. when more than two CSP's systems are inaccessible. As described above, the 'Normal' file priority level stores 48 parity information and hence cannot be able to recover data corruption of more than 48 shards. Hence as depicted in figure 5.16 the file recovery process failed for files uploaded using the 'Normal' priority level with more than 48 corrupted shards.

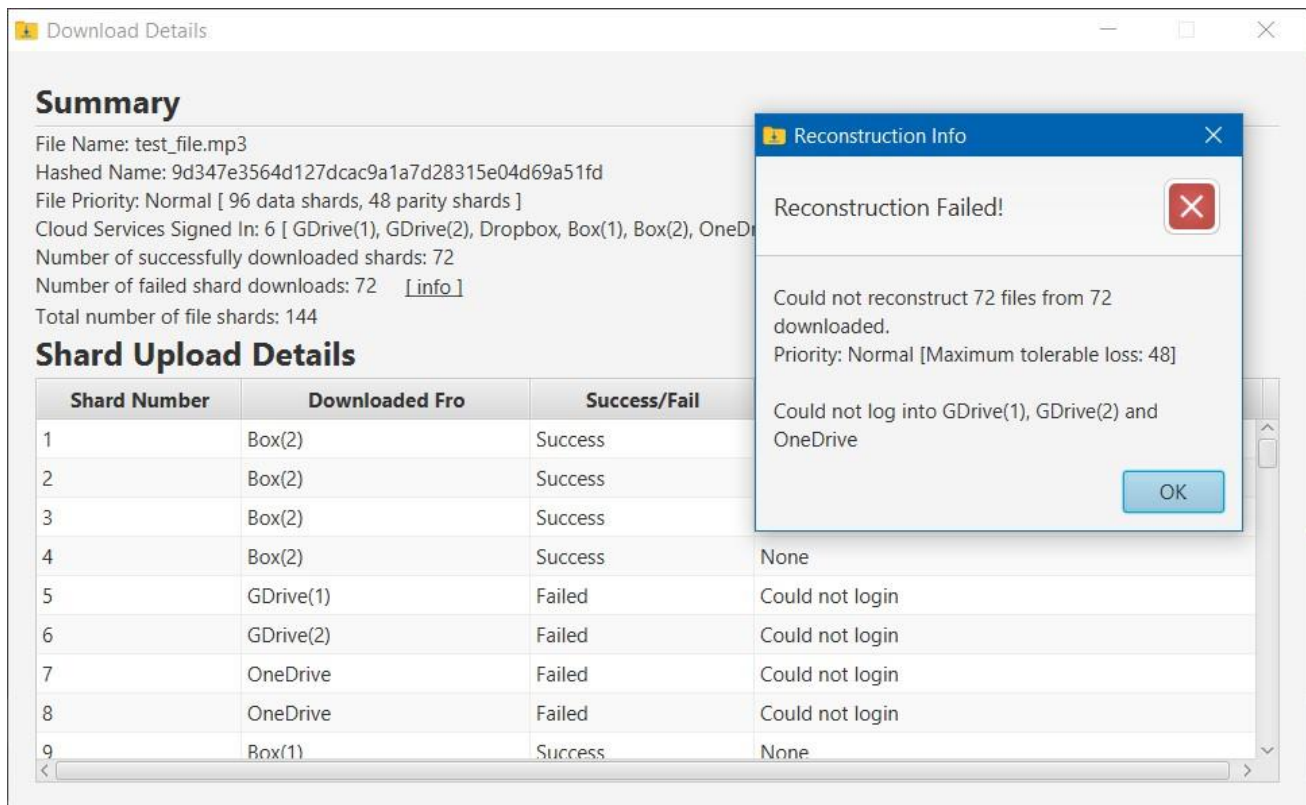


Figure 5.16 – failed file reconstruction during download for more than 48 corrupted shards with Normal option

Scenario 2: Downloading a file by choosing the ‘Important’ file priority level.

Case 1

The ‘**Important**’ priority level when selected splits a file into 72 data shards and 72 parity shards that are used for file recovery in the event of corruption making a total of 144 shards that are uploaded to the six CSP’s using the data dispersal technique. Although slower in recovering a file than the ‘Normal’ priority level, the ‘Important’ priority level enables a file to be recovered even if 72 shards are corrupted or when three CSP’s servers are inaccessible.

Figure 5.17 depicts a scenario where 72 of the distributed shards are corrupted or a situation where three of the six CSP’s servers are unreachable but the SMF system still recovered the full file during download.

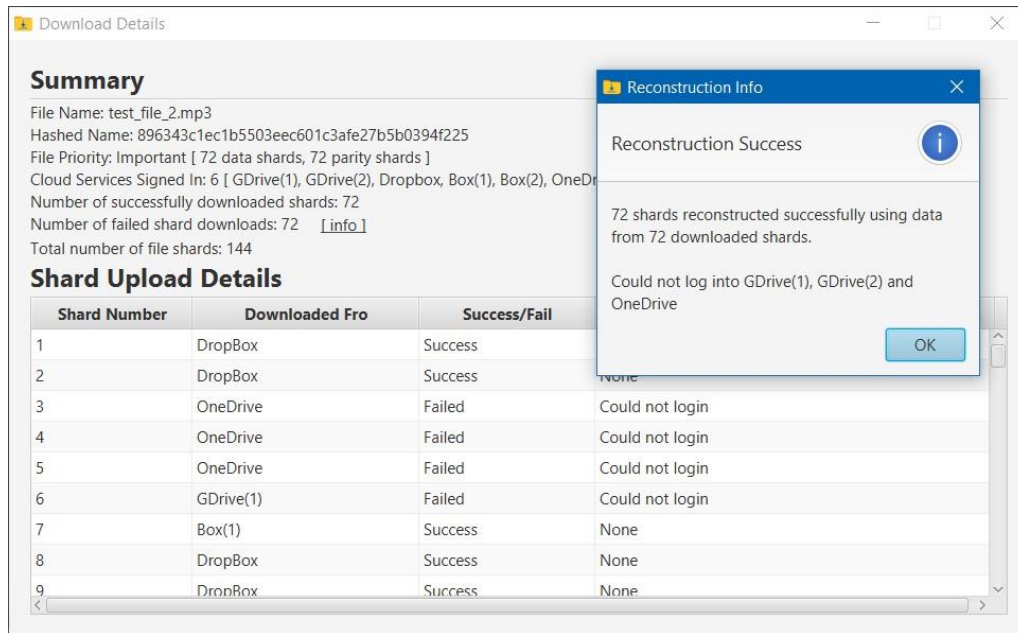


Figure 5.17 – successful file reconstruction during download for 72 corrupted shards with Important option

Case 2

Case 2 presents a scenario that depicts a situation where more than 72 shards are corrupted i.e. when more than three of the CSP's systems are inaccessible. As stated above, the 'Important' file priority level stores 72 parity information and hence cannot recover data corruption of more than 72 shards. Therefore, as depicted in figure 5.18 the file recovery process failed.

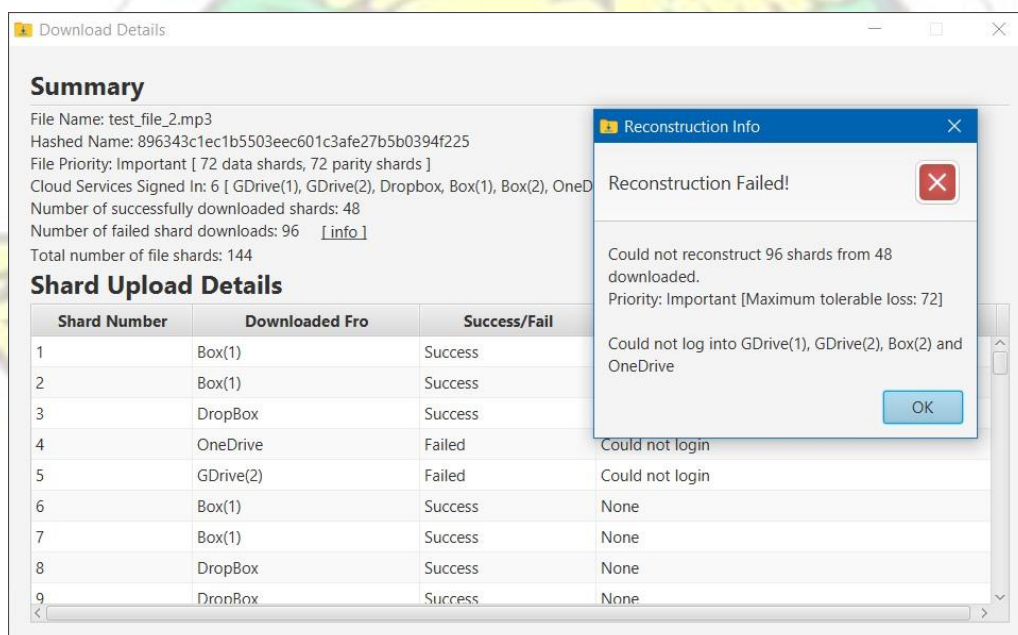


Figure 5.18 – failed file reconstruction during download for more than 72 corrupted shards with Important option

Scenario 3: Downloading a file by choosing the ‘Critical’ file priority level.

Case 1

Figures 5.19, 5.20, and 5.21 respectively depict a scenario where 4, 8 and 12 of the distributed shards are corrupted during download and the SMF system reconstructs the file successfully with Critical option.

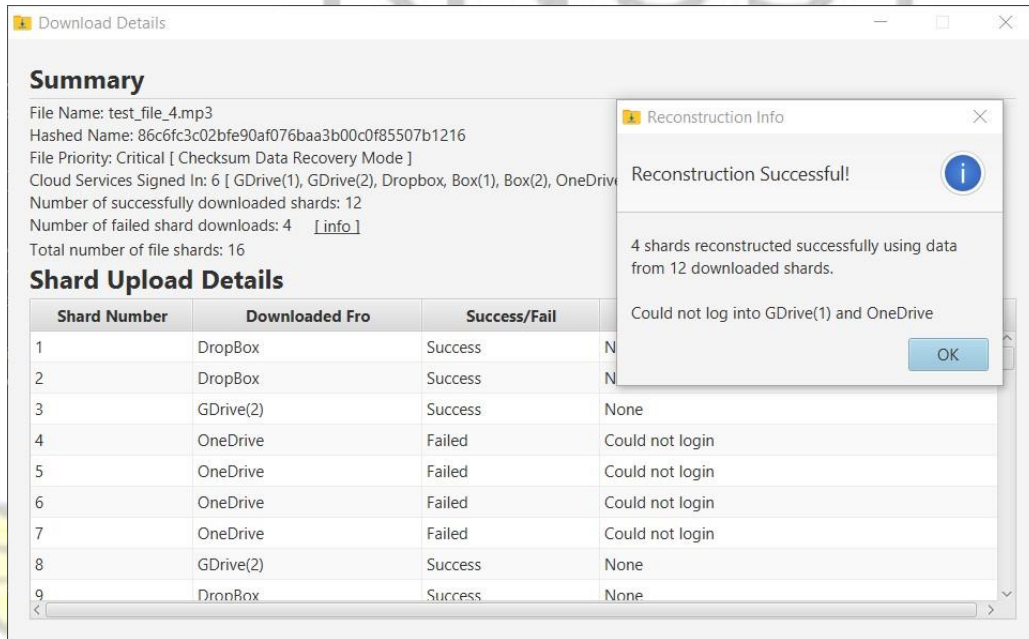


Figure 5.19 – successful file reconstruction during download for 4 corrupted shards with Critical option

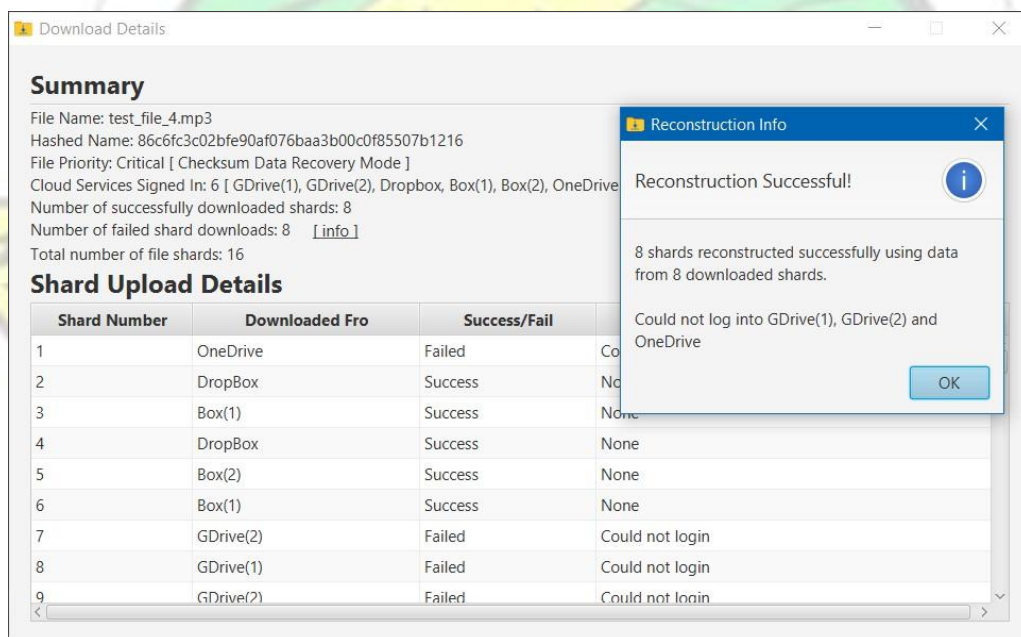


Figure 5.20 – successful file reconstruction during download for 8 corrupted shards with Critical option

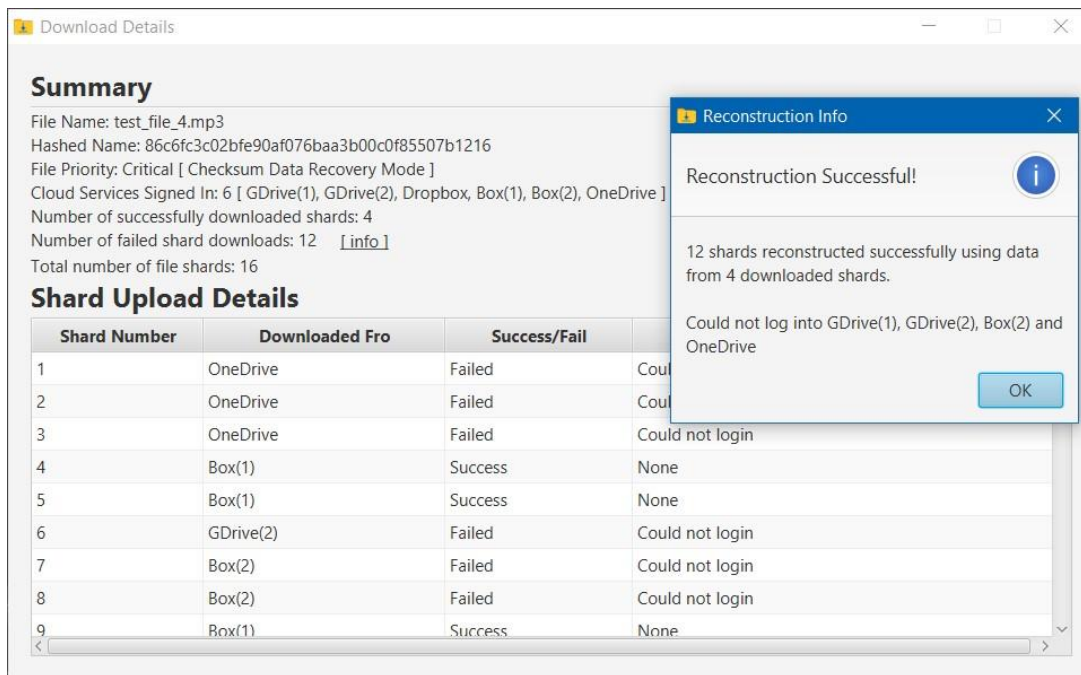


Figure 5.21 – successful file reconstruction during download for 12 corrupted shards with Critical option

Case 2

It was realised during testing as shown by Figure 5.22 that the ‘Critical’ priority option cannot recover a file when more than 12 of the shards are corrupted.

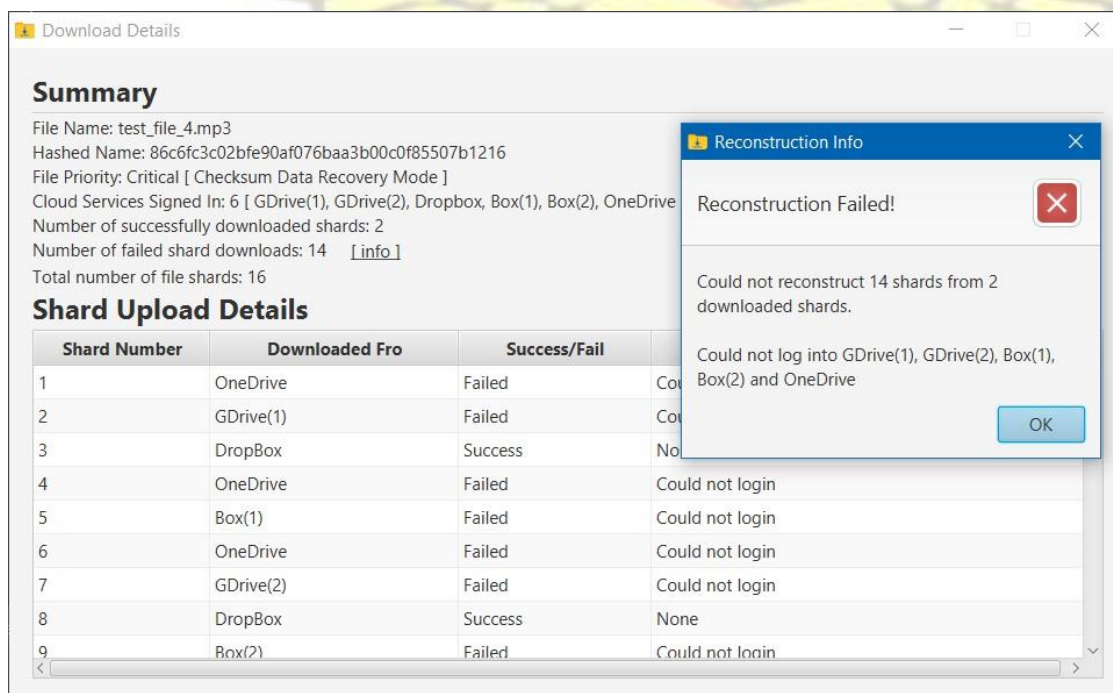


Figure 5.22 – failed file reconstruction during download for more than 12 corrupted shards with Critical option

5.4. The Proposed Cloud Data Distribution Intermediary (CDDI) Framework

The study proposes a Six-level Cloud Data Distribution Intermediary (CDDI) Framework that addresses the study objectives as shown in Figure 5.12

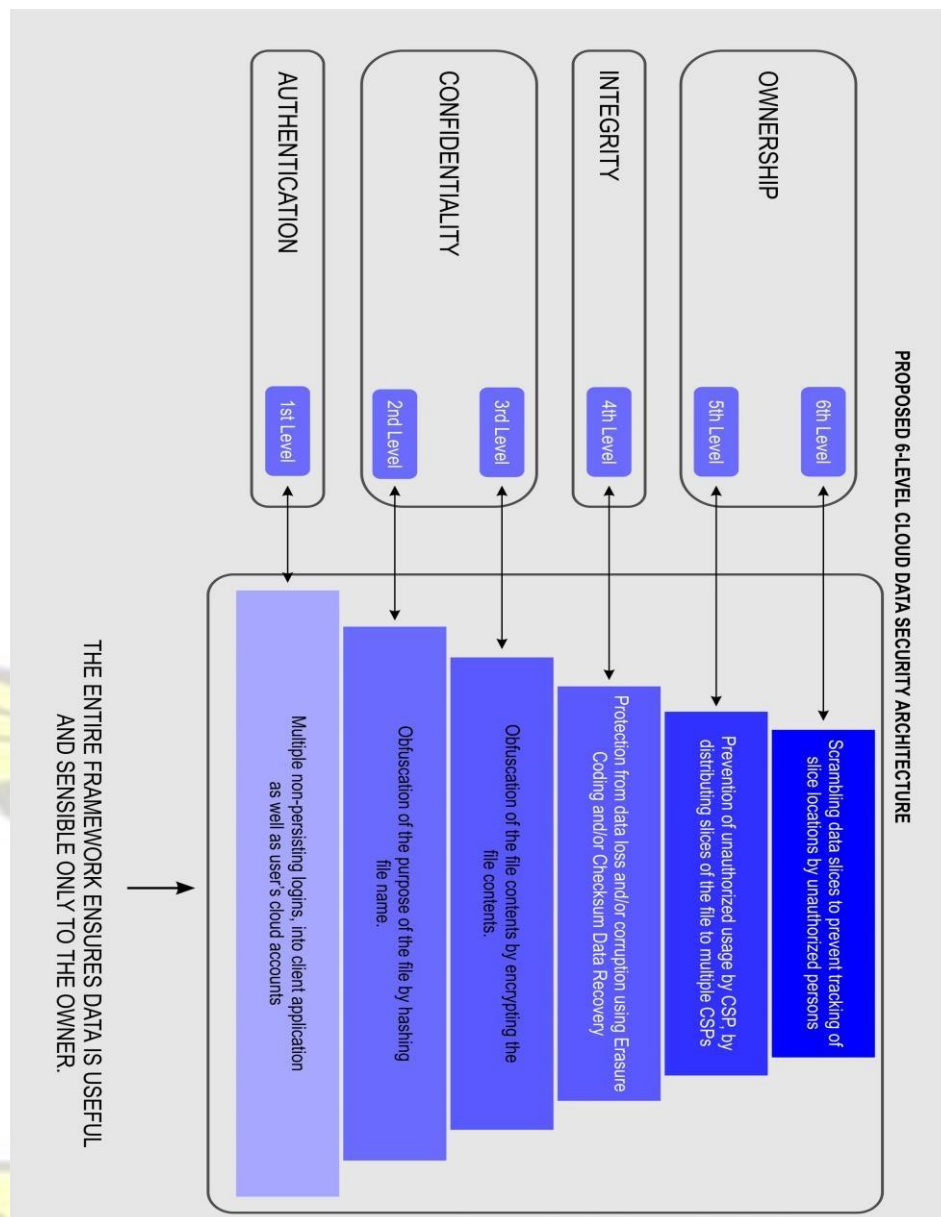


Figure 5.23 – Proposed Six-level Cloud Data Distribution Intermediary (CDDI) Framework

5.5. Discussion of how the proposed system compares with existing related systems

Below is a summary of how the proposed system compares with other existing related systems in terms of architecture, ensuring confidentiality, providing integrity, and controlling access to data and other resources outsourced for cloud storage.

5.5.1. Architecture

The Google File System (GFS) and Apache Hadoop are distributed file storage systems whereas Backblaze offers cloud storage and backup service (Refer to Section 2.9). The proposed system is a distributed cloud backup service. GFS is Google's proprietary distributed file storage system upon which the Google Drive cloud storage is built.

GFS and Apache Hadoop make use of clusters of commodity machines for data storage and computations (Hadoop 2013; Roshan 2014). Backblaze utilizes a single data centre to hold all of the backed-up data (Backblaze, 2017). However, the proposed system makes use of multiple existing cloud storage service providers such as Google Drive, Dropbox and Box, to store pieces or shards of a single file.

5.5.2. Confidentiality

Google File System and Apache Hadoop are designed to support constant data access by applications that perform computations with the stored data. As such the recommendation is for the data to be plain or raw (Hadoop, 2013; Roshan, 2014). Any external party that accesses the data is able to get the data in its plain format. While this is not a strict requirement, it is the recommended design. Confidentiality is traded off for computational ease in these systems.

Backblaze uses a combination of Advanced Encryption Standard (AES) and Secure Socket Layer (SSL) to secure the data that is transmitted to and saved on their servers, thus ensuring confidentiality (Backblaze, 2017).

The proposed system uses a custom-made encryption algorithm based on the motions of a Rubik's Cube to obfuscate the data. In addition, the filename is also hashed to obscure the purpose of the file. The combination of the data encryption and filename hashing greatly enhances the confidentiality of the system.

5.5.3. Access Control

Google File System, Apache Hadoop and Backblaze all store data shards in a location where the service provider has access to all the pieces of the file data. As such, while it may be possible that the data owner may be unaware of the storage location of their file shards, the service providers have all that information available to them (Chima, 2016). On the other hand, the proposed system distributes the data shards to multiple service providers without storing the credentials required to access them.

This means that no one service provider knows the location of all the file shards, ensuring that only the data owner has full access to the data.

To access file shards on Apache Hadoop and Backblaze, the data owner must supply a single set of login credentials. The service providers require no login credentials to access the files saved on their servers and since all the file shards are hosted on their storage servers, they have full access to the files that are uploaded to their servers.

In contrast, with the proposed system requires a separate set of login credentials for each of the cloud storage service providers that the user subscribes to. Any user or service provider that wishes to access files from the proposed system will need to know at least all but one of the login credentials.

This greatly increases the confidentiality factor of the storage framework.

5.5.4. Integrity

The GFS and HDFS file systems are designed for appending to files but not altering the file contents. This helps to prevent altering the content of the files which are saved on the file system, thus securing the integrity of the data.

On the other hand, Backblaze uses Reed-Solomon erasure coding via Vandermonde matrix to guard against data loss. While this algorithm protects against data loss when a storage cabinet goes offline, it does nothing to prevent alteration of the data in a shard. As long as the shard is present, it is included in the downloaded file. This means that the data in the shard can be altered without detection by the data owner. This observation was noted through compiling and running the Backblaze open source Reed Solomon Erasure Coding Source Code (Backblaze, 2015b)

The proposed SMF system employs Reed-Solomon, and the proposed checksum data recovery method that are implemented at the client side for error detection and correction. The SMF client application ensures that any alterations to the data can be detected and corrected.

5.5.5. Ownership

The SMF system ensures that only the owner of the data (cloud subscriber) has sole ownership of their data resource stored on the cloud. This is in contrast to existing architectures (direct or indirect models) implemented by storage CSP's such as Google Drive, Dropbox, or Box where ownership of the data becomes a contentious issue but in most cases the CSP claim ownership (Gray, 2014; FileCloud, 2016).

CHAPTER 6

FINDINGS, CONCLUSIONS AND RECOMMENDATIONS

6.0. Findings

The finding from the study on how it addresses issues of cloud data confidentiality, integrity, ownership, availability and authentication is presented by Tables 6.1. In addition Table 6.2 presents how the CDDI framework addresses other cloud security issues as Multi-Tenancy, Data Loss, Data Location and other computer network attacks such as Dos/DDos, Malicious Insider, Malware Injection, Man-in-the-middle (MITM), Message Replay and U2R and R2U.

Table 6.1 - How the proposed system address issues of Confidentiality, Integrity, Ownership, Availability and Authentication

Confidentiality	From figure 5.3 and figure 5.4 the CDDI framework provides confidentiality of the subscriber's data at the second level (through obfuscation of the purpose of the file by hashing the file name) and third level (through obfuscation of the file content by encrypting the file content)
Integrity	The CDDI provides protection from data loss or corruption using the Reed Solomon Coding or the Checksum Data Recovery depending on the subscribers file priority level selected. Data integrity is achieved at the fourth level of the CDDI framework
Ownership and Availability	The CDDI ensures that the cloud subscriber has sole ownership of their data outsourced for cloud storage at levels five and six of the framework. In addition, the same levels of the framework together with the use of the metadata ensure that unavailability of a CSP that may be as a result of a DoS/DDoS attack does not prevent the subscriber from having access to their data. Thereby assuring subscribers the availability of their data.
Authentication	The CDDI uses multiple non-persistent logins to access the client interface as well as the subscriber's cloud accounts. The use of non-persistent logins means that even if a

	malicious person gains knowledge of one of the credentials, the person's ignorance of the remaining credentials will serve as a check to prevent access to the subscriber's data. Level one of the framework ensures this security feature.
--	---

Table 6.2 - How the CDDI framework addresses other cloud security issues

Multi-Tenancy	<p>The CDDI framework addresses the issue of multi-tenancy threat by the use of the different CSP's storage facilities. The framework by breaking and distributing shards to multiple</p> <p>CSP's means that, the cloud security challenge of multi-tenancy which poses threat of a CSP maliciously leaking subscriber's data to a competitor deliberately or accidentally is eliminated.</p> <p>The CSP has no access to the subscriber's complete data as only a portion of the data is stored with them.</p>
Data Loss	<p>The CDDI framework prevents Data Loss (Erasure) through using the parity information stored on the metadata server and the use of the File Decoding Process via Reed Solomon Decoding method or the Checksum Data Recovery method, depending on the user's choice of a priority level during the File Upload. The proposed framework compared to the existing cloud file architectures (google GFS, Apache Hadoop, and Backblaze B2 – Refer to Section 2.9 of the literature review) can recover the most data. Also the CDDI framework unlike the Backblaze B2 system is able to detect if an attacker alters the content of a shard and maintains the shard size.</p>
Data Location	<p>The current direct or indirect cloud architectures gives the provider access to the subscribers data as they know the locations of their storage facilities and can have access to them to retrieve the data even if encrypted (Chima, 2016). The CDDI framework distributing shards to multiple CSP's (with storage facilities located in different countries) prevents a single CSP from knowing the location of the subscriber's data and thereby</p>

	addressing the issue of different privacy laws of different countries.
DoS/DDoS Attack	The CDDI framework distributing the data to different multiple CSP's storage facilities means that no single CSP has the subscriber's complete data. Hence a DoS/DDoS attack on one or more CSP's does not prevent the subscriber from accessing their outsourced data. The parity information stored on the CDDI metadata server together with the File downloading process can be used to recover the data even if several CSP's are attacked.
Malicious Insider Attack	By splitting subscribers data into shards and distributing to different multiple CSP's storage infrastructures, the CDDI framework protects subscribers data against an insider attack as an employee of a CSP only have access to scrambled portions of the subscribers data. The CDDI framework ensures that only the rightful owner of the data can make use of the data.
Malware Injection Attack	The CDDI framework addresses the threat of cloud malware injection attack where the attacker plant an evil virtual cloud machine in a CSP's cloud environment with the goal of intercepting subscribers data and taking full control. The framework using its metadata information about location of shards stored on the metadata server and the File downloading process can track and restore corrupted shards that may have been altered by the Malware Injection attacker. Also as the data received by the attacker is incomplete the attacker cannot make use of the data. In the event of this attack occurring, the CDDI framework treats the data sent to the evil cloud virtual machine as lost and recover using the metadata and either the Reed Solomon Decoding method or the Checksum Data Recovery method depending on the priority selected for the upload.
MITM Attack and Message Replay Attack	The CDDI framework distributing the split shards to different CSP's infrastructures minimises the threat of MITM attack in the sense that the attacker will have to intercept all of the

	distributed file splits for the MITM attack to be effective. Since the shards are distributed to different multiple CSP's the intercepted data will be incomplete and un-useful to the attacker. Even if the attacker commits a Message Replay attack by changing the content of the intercepted shards, the CDDI framework metadata server can be used with the File Download Process to reconstruct the file to its original form.
U2R and R2U attacks	The U2R attack enables attacker to maliciously log into a system as a legitimate user using authorised system credentials and R2U attack enables an attacker to exploit a system vulnerabilities through sending probing packets to the system. The CDDI framework addresses threats from these attacks through the use of different multiple cloud storage facilities to store the distributed fragments of the subscriber's data. No single CSP has the subscriber's complete data and hence a successful U2R or R2U attacker only sees a portion of the subscribers data which will be scrambled and un-useful.

The study also compared the proposed architecture with existing architectures and the findings are as presented in Table 6.3

Table 6.3 – Comparison of the CDDI framework with existing architectures

	Direct Model	Indirect (CASB) Model	Indirect (CDDI) Model
Architecture	The subscriber places a file into the CSP's interface for onward processing and uploads to the Cloud.	The CASB monitors all data transfers within the organization as well as the transfer of files out of (and into) the organization. All transfers are therefore filtered by the CASB. (Rubens, 2017)	The CDDI handles interactions with the CSPs on behalf of the subscriber. The subscriber is required to have a minimum of 6 Cloud Storage accounts. The CDDI

			performs data obfuscation on behalf of the subscriber.
Data Privacy	<p>The CSP is responsible for ensuring the privacy of the subscriber's data. The method used to encrypt the file is known to the CSP, as well as the key or manner in which the key is generated. As such, if the CSP so desires, they may decrypt the file for their personal purposes (TipTopSecurity, 2016).</p>	<p>The CASB prevents unauthorized access to the organization's confidential data by preventing the confidential data from ever being transferred to the cloud. The CASB uses machine learning to determine data transfers that infringe the organization's regulations, and then halts the transfer.</p>	<p>In the proposed model, the CDDI allows all transfers of data to the cloud, but first encrypts the data locally (outside the CSPs reach) then splits the data into a number of shards, and randomly distributes the shards to multiple CSPs. The number of shards received by each CSP is insufficient to reconstruct the original file. This way, the file remains confidential and useful to only the owner.</p>
Unauthorized Data Use	<p>The CSP has access to the entire data and how to decrypt it. As such the CSP can use the data for any purpose without notifying the owner or requiring the owner's permission (Chima, 2016).</p>	<p>The CSP has access to the organization's non-critical data (data not captured in the organization's privacy regulations). The CSP is able to decrypt this data to use as they please.</p>	<p>The CSP has access only to incomplete and encrypted portions of the data. Without the other portions of the data, it is very difficult to decrypt as a result of the encryption algorithm used. Thus the CSP is prevented from accessing the data</p>

			for their own purposes.
Unauthorized Data Access	One set of credentials are required to gain access to the data. Anyone with this single set of credentials can access the entire data.	The CASB serves as a proxy that also filters the traffic moving into and out of the organization. Thus the only credentials needed to access the data on the cloud, is the login credentials for the cloud account. Any individual with the login credentials, therefore, has access to the organization's data.	The CDDI is designed to demand login credentials of the system (SMF). Further, to access the data stored on the cloud, each cloud account must be signed into individually. Thus requiring multiple authentications before access and usage.
Data Ownership	Unless the client applies encryption before sending the data to the Cloud Service Provider, the provider can claim full ownership of the data as they have full access and control over it (FileCloud, 2016).	The CASB may encrypt the data before forwarding to the Cloud Service Provider, to ensure that the data is safe on the cloud.	The CDDI encrypts the data at the SMF client side and splits the data before distribution to the CSPs. Since the data is sent to multiple CSPs, no individual CSP can claim ownership of the complete data, except the data owner.
Data Integrity	The CSP is responsible for ensuring the integrity of the data. Most CSPs provide version control services which allow the subscriber to revert to a previous version of the file if the current version is damaged or	The CASB turn over responsibility of ensuring data integrity to the CSP upon the upload of the data. As such the subscriber has access to previous versions of the file but also suffers in the	The CDDI uses ReedSolomon Coding as well as the proposed Checksum Error Detection and Correction Program to verify the integrity of data and perform error

	<p>otherwise modified.</p> <p>However, a malicious insider within the CSP may delete all traces of the file, making it irrecoverable (TipTopSecurity, 2016).</p>	<p>event of a malicious insider attack.</p>	<p>corrections in the event that portions of the data get corrupted. The subscriber however does not get access to previous versions of the file but is protected from malicious insider attacks that happen in one or more CSPs.</p>
Data Availability	<p>The data is available anytime the CSP is in operation.</p> <p>However, in the event of a DoS/DDoS attack on the CSP, the subscriber has no access to the data.</p>	<p>The CASB relies on the CSP to ensure availability of the subscriber's data. (Rubens, 2017)</p>	<p>In the event of DoS/DDoS attack on any of the CSPs, the remaining data that is stored on the other CSPs can be used to reconstruct the original file. Hence the subscriber is insured and assured of data availability even when some of the CSPs are offline.</p>

6.0.1. Novelty of the Findings

The SMF system with its characteristics provides solutions to the cloud data security challenges outlined by this study. The system is unique as no single product of its kind was found in the market. All the existing cloud security solution systems either secures subscribers data on a single CSP's infrastructures (Direct model) or employ the service of SECaaS provider through setting up regulations and policies via a CASB.

6.0.2. Contributions to knowledge

The following are the studies major contributions to knowledge:

The proposed advanced detection and correction solution can recover any number of data errors or deletions as compared to the existing implementations based on the Reed Solomon coding that are capable of recovering up to the number of parity bits for deletions and up to only half the number of parity bits for errors.

The approach proposed for the generation of the encoding polynomial coefficients is much efficient than the existing.

The proposed transposition cipher algorithm based on Rubiks cube transformation is much stronger and difficult to decipher making the proposed cloud data security solution very secure. This together with the other techniques used as data dispersal, and data shuffling, will eliminate the subscriber fear of using cloud computing and enjoying its numerous benefits stated in this thesis.

Three articles have been published from the study in peer reviewed journals (See Appendix 3 for the articles and also the reviewer's comments).

6.1. Conclusions

This study's main purpose was to address the security challenges in relation to outsourcing data and other resources for third party Cloud Service Providers storage particularly in terms of preventing the CSP from making use of the data. The study purpose has been achieved as the proposed system (See Appendix 2) is able to assure of the confidentiality, integrity, and as well able to effectively control and manage who has access to the data and can make use of it.

This section discusses how the proposed system addresses the problem statement and the objectives of the study as outlined below.

6.1.1. How can we ensure data outsourced for cloud storage is only useful to only the data owner?

In computer security, the CIA trade notes three security dimensions as Confidentiality, Integrity and Availability. The proposed system uses a combination of hashing, encryption, scrambling, erasure coding and data dispersal (Refer to Chapter 4 Section 4.3) to address these security dimensions and also addresses cloud subscribers concerns of data Ownership, data Usage, data Location, and other security issues that poses threats to data outsourced for cloud storage.

6.1.2. The proposed Cloud Data Security Solution Framework

The proposed CDDI framework (Figure 5.23 in Chapter 5) achieves the study's purpose of alleviating the cloud subscriber fear of their data privacy, unauthorized usage of their data, the unknown location of their data and ownership of the data outsource for cloud storage

6.1.2.1. How can cloud data be secured to prevent unauthorized access?

The CDDI framework when implemented distributed shards to multiple CSPs. The slices of data which each CSP receives from the data dispersal technique of the file uploading sequence (Refer to Section 5.2.5) are both incomplete and encrypted. This means that the CSP doesn't have access to the subscriber's full data. Only the cloud subscriber has access to the full data through applying the CDDI file download sequence described in Section 5.3.

6.1.2.2. In what ways can a cloud service subscriber prevent their data from being used for other purposes by the cloud provider?

The CSP's are unable to make use of the data entrusted in their care as they receive incomplete and encrypted slices of the data when the CDDI framework is employed. They can only store the data but cannot use it for any other purpose (See Figure 5.9 - Content of Dropbox and Figure 5.10 - Content of Box).

6.1.2.3. In what ways can the cloud subscriber ensure that their outsourced data is not vulnerable as a result of the data location since different countries have different data privacy laws?

Unlike the current cloud data storage where the subscriber's data is vulnerable as it resides with only one provider (Chima, 2016), the CDDI framework randomly disperses slices of the resource to multiple CSPs. This prevents the CSPs from having access to all the files as well as guessing the locations of the slices which they do not have. Hence in countries where the data privacy laws are liberal or not strictly enforced, the CSP is still unable make use of the portions of the data entrusted with them.

6.1.2.4. In what ways can the cloud subscriber ensure that they have sole ownership of their data outsourced for cloud storage?

The scrambling and data dispersal features of the CDDI framework enforce single-ownership of the data. The system ensures that the file is never whole and useful anywhere except on the original user's computer.

6.1.3. Comparison of Proposed Architecture with Existing Architectures

The proposed CDDI framework was compared with existing architectures (i.e. the direct model and the indirect (CASB) model) in relation to how it secures subscribers data against Data Privacy, Unauthorized Data Use, Unauthorized Data Access, Data Ownership, Data Integrity, and the Architecture itself.

In relation to Data Privacy, the framework encrypts data locally on the subscribers' system and split the data into shards and randomly distributes them to different CSP storage facilities. Since a CSP has access to only a portion of the subscribers' data, this also enforces issues of Data Ownership, unauthorized Data Access and Unauthorized Data Usage.

In addition, the framework uses the Reed-Solomon Coding and the Proposed Checksum Error Detection to check for Data Integrity.

Finally, in relation to architecture, the CDDI framework takes care of interactions with the CSPs thereby interfacing between the subscriber system and the CSP and hence providing security.

6.2. Recommendations

It is recommended the system is commercialised and be used by individuals or organisations considering migrating their resources for cloud storage but are sceptical of security of the resource. The proposed cloud data security solution is especially useful for environment where the security of these resources is vital and cannot be compromised such as Financial Industry, Health Data, and Insurance Policy Documents.

Based on the results that will be obtained from the commercial usage of the system, further research will be conducted on the system's implementation resource requirements, performance and security.

REFERENCES

- ABS (2016). ABS Update – 2016 Online Census Form. [Online]. Available from: <http://www.abs.gov.au/ausstats/abs@.nsf/mediareleasesbyReleaseDate/617D51FA32D27BF9CA25800A0077B7BD?OpenDocument> [Accessed: 23rd October 2017].
- Ahmed, N. (2017), "Cloud Computing: Technology, Security Issues and Solutions", *IEEE*, [Online]. Available from: <http://ieeexplore.ieee.org/document/7905258/> [Accessed: 20th October 2017].
- Ahmed, M., Hossian, M. A., (2014), "Cloud Computing and Security Issus in the Cloud", *International Journal of Network Secuity and its Applications*. Vol. 6, No. 1, pp. 25-36.

- Alani, M. M. (2016), "Security Attacks in Cloud Computing", *Elements of Cloud Computing*, Springer Brief in Computer Science, pp. 41-50 https://link.springer.com/chapter/10.1007/978-3-31941411-9_4
- Ali, M., Khan, S. U., Vasilakos, A. V. (2015), "Security in cloud computing: opportunities and challenges", *Information Sciences*, Vol. 305, No. 1, pp. 357-383.
- Amazon Web Services, (2010). *Software-as-a-Service (SaaS) on AWS*. [Online]. Available from: https://d36cz9buwru1tt.cloudfront.net/SaaS_whitepaper.pdf [Accessed: 10th February 2014].
- Apache Hadoop, (2014). What is Apache Hadoop? [Online]. Available from: <http://hadoop.apache.org/> [Accessed: 15th June, 2014]
- BackBlaze, (2015a). Backblaze Open Sources Reed-Solomon Erasure Coding Source Code. [Online]. Available from: <https://www.backblaze.com/blog/reed-solomon/> [Accessed: 11th January, 2016]
- BackBlaze, (2015b). Backblaze Open Sources Reed-Solomon Erasure Coding Source Code. [Online]. Available from: <https://www.backblaze.com/blog/vault-cloud-storage-architecture/> [Accessed: 11th January, 2016]
- BackBlaze, (2017). Cloud Storage that's astonishingly easy and low-cost. [Online]. Available from: <https://www.backblaze.com/> [Accessed: 20th March, 2017]
- Barry, D. K. (2000). *Service Architecture (SaaS)*. [Online]. Available from: http://www.servicearchitecture.com/articles/cloud-computing/software_as_a_service_saas.html [Accessed: 14th July 2016]
- Benvenuto, C. J. (2012). Galois Field in Cryptography. [Online]. Available from: https://www.math.washington.edu/~morrow/336_12/papers/juan.pdf [Accessed: 14th May 2016]
- Carson, C. (2016). How much data does Google store? [Online]. Available from: <https://www.cirrusinsight.com/blog/much-data-google-store> [Accessed: 20th February, 2016]
- Chauhan, K. (2015). "Ensuing Data Storage Security in Cloud Computing", *International Journal of Computer Science and Information Technology Research*, Vol. 3, No. 2, pp. 283-287
- Chima, R. (2016). Cloud Security – Who owns the data? [Online]. Available from: <https://www.bbconsult.co.uk/blog/cloud-security-who-owns-the-data> [Accessed: 20th February, 2016]
- Chong, F., Carraro, G., and Wolter, R. (2006). *Multi-Tenant Data Architecture*. [Online]. Available from: <https://msdn.microsoft.com/en-us/library/aa479086.aspx> [Accessed: 12th July, 2016]
- Chou, Te-Shun (2013), "Security Threats on Cloud Computing Vulnerabilities", *International Journal of Computer Science and Information Technology*, Vol. 5, No. 3, pp. 79-88.
- Cox, R. (2012). Finite Field Arithmetic and Reed-Solomon Coding. Available from: <http://research.swtch.com/field> [Accessed: 14th May 2016]

cs.cmu.edu (1998). Reed-Solomon Codes. An introduction to Reed-Solomon codes: principles, architecture and implementation. Available from:

https://www.cs.cmu.edu/~guyb/realworld/reedsolomon/reed_solomon_codes.html

[Accessed: 14th May 2016]

CSA (2011). Security guidance for critical areas of focus in cloud computing V3.0. [Online]

Available from: <https://cloudsecurityalliance.org/guidance/csaguide.v3.0.pdf> [Accessed: 10th May, 2015]

CSA (2009). Security guidance for critical areas of focus in cloud computing V2.1. [Online] Available from: <https://cloudsecurityalliance.org/csaguide.pdf> [Accessed: 10th May, 2015]

Czynszak, S., (2011). Decoding algorithms of Reed-Solomon Code. Available from: <http://www.diva-portal.org/smash/get/diva2:833161/FULLTEXT01.pdf> [Accessed: 10th May, 2015]

Dell Power Solutions, (2005). *Enhancing High-Performance Computing Clusters with Parallel File Systems*. [Online]. Available from: <http://www.dell.com/downloads/global/power/ps2q05-20040179Saify-OE.pdf> [Accessed: 14th May 2014]

DeZyre, (2016). Hadoop Architecture Explained – What it is and why it matters. [Online]. Available from: <https://www.dezyre.com/article/hadoop-architecture-explained-what-it-is-and-why-it-matters/317> [Accessed: 15th March, 2017]

DoubleHorn,(2017). Cloud Services Brokers: The future of SaaS and IaaS Consumption [Online]. Available from: <https://doublehorn.com/cloud-services-brokers-the-future/> [Accessed: 15th October, 2017]

Dropbox, (No Date). Under the hood: Architecture Overview. [Online]. Available from: <https://www.dropbox.com/business/trust/security/architecture> [Accessed: 1st Nov., 2017]

Educause, (2009). 7 Things you should know about cloud computing. [Online]. Available from: <https://library.educause.edu/~media/files/library/2009/8/est0902-pdf.pdf> [Accessed: 14th May 2014]

Fahmida Y. R. (2016). The dirty dozen: 12 cloud security threats. [Online]. Available from: <https://www.infoworld.com/article/3041078/security/the-dirty-dozen-12-cloud-security-threats.html> [Accessed: 15th March, 2017]

FileCloud, (2016). Data Ownership in the Cloud – How does it affect you? [Online]. Available from: https://www.getfilecloud.com/blog/2016/11/data-ownership-in-the-cloud-how-does-it-affectyou/#.WgG7_I-0Pct [Accessed: 15th March, 2017]

Forcepoint, (2017). How Forcepoint Web Security Cloud Works. [Online]. Available from: https://www.websense.com/content/support/library/web/hosted/getting_started/cws_explain.aspx [Accessed: 15th Oct., 2017]

Forouzan, B. A. (2001). Data Communications and Networking. (2th edn). McGraw-Hill. ISBN: 0072822945

Freach, J. (2011). The art of design research (and why it matters). [Online]. Available from: <https://www.theatlantic.com/entertainment/archive/2011/05/the-art-of-design-research-and-why-it-matters/239561/> [Accessed: 25th Oct., 2017]

Goswami, B. and Singh, S. N. (2012), "Enhancing security in Cloud Computing using Public Key Cryptography with Matrices", *International Journal of engineering Research and Applications*, Vol. 2, No. 4, pp. 339- 344.

Gray, D. (2014). Data ownership in the cloud. [Online]. Available from: <http://dataconomy.com/2014/03/data-ownership-in-the-cloud/> [Accessed: 13th October, 2017]

Hadoop, (2013). HDFS Architecture Guide [Online]. Available from: https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html [Accessed: 15th June, 2014]

Haiman, M. (Date). Notes on Reed Solomon. Available from: <https://math.berkeley.edu/~mhaiman/math55/reed-solomon.pdf> [Accessed: 13th December, 2014]

Hansche, S., Berti, J., and Hare, C. (2013). Chapter 6: Cryptography. [Online]. Available from: <http://www.crcnetbase.com/doi/abs/10.1201/9780203507872.ch6> [Accessed: 13th December, 2014]

Higashi, M. (2014). 3 Threats to Cloud Data, and How to address them. [Online]. Available from: <https://ciphercloud.com/3-threats-cloud-data-security-address/> [Accessed: 13th October, 2017]

Hill, T. (2013). Reed Solomon Codes Explained. [Online]. Available from: <https://www.tonyhill.info/app/download/.../Reed+Solomon+Explained+V1-0.pdf> [Accessed: 13th December, 2014]

Hussain, S. A., Fatima, M., Atif, S., Imran, R., Raja, K. S., (2017). "Multilevel classification of security concerns in Cloud Computing", *Applied Computing and Informatics*, Vol. 13, pp. 57-65

Huth, A. and Cebula, J. (2011). *The basics of cloud computing*. [Online] Available from: <https://www.us-cert.gov/sites/default/files/publications/CloudComputingHuthCebula.pdf> [Accessed: 28th Aug., 2015]

InfoSec, (2017). Cloud Computing Attack Vectors and Counter Measures. [Online]. Available from: <http://resources.infosecinstitute.com/cloud-computing-attacks-vectors-and-counter-measures/#gref> [Accessed: 1st May, 2017]

Jain, R. (2013). Hadoop and HDFS for Beginners. [Online]. Available from: <https://www.slideshare.net/rahuldausa/hadoop-hdfs-for-beginners> [Accessed: 15th June, 2014]

Kessler, G. C. (2017). An overview of Cryptography. [Online]. Available from: <http://www.garykessler.net/library/crypto.html> [Accessed: 1st May, 2017]

- Khan, N., Yasiri, A. (2016), "Identifying Cloud Security Threats to Strengthen Cloud Computing Adoption Framework ", *Procedia Computer Science, ScienceDirect*, Vol. 94, pp. 485-490.
- Khandelwal, S. (2017). It's 3 Billion! Yes, Every Single Yahoo Account Was Hacked In 2013 Data Breach. Available from: <https://thehackernews.com/2017/10/yahoo-email-hacked.html> [Accessed: 1st May, 2017]
- Kharche, H., and Chouhan, D. S. (2012), "Building Trust in Cloud Using Public Key Infrastructure", *International Journal of Advanced Computer Science and Applications*, Vol. 3, No.3, pp. 26-31.
- Khatri, S. K., Singhal, H., Bahri, K. (2013), "Multi-Tenancy Engineering Architecture in SaaS", *International Journal of Computer Applications*. [Online]. Available From: <http://research.ijcaonline.org/icrito/number1/icrito1309.pdf> [Accessed: 10th October, 2016]
- Krishna, B. H., Kiran, S., Murali, G., Reddy, R., (2016). "Security Issues in Service Model Of Cloud Computing Environment", *Procedia Computer Science*, Vol. 87, pp. 246-251
- Laudon, K. C. and Laudon, J. P. (2010). *Management Information System*. (11th edn). New Jersey: Pearson Edition, Inc.
- Lee, P. (2012). Design Research: What is it? Why do it? [Online]. Available from: <https://reboot.org/2012/02/19/design-research-what-is-it-and-why-do-it/> [Accessed: 25th October, 2017]
- Lynson, B. Tutorial on Reed-Solomon Error Correction Coding. NASA Tech Brief MSC-21834. Available from: <http://jeffareid.net/misc/msc-21834.pdf> [Accessed: 1st May, 2017]
- Lui, S. (2017). #CensusFail 2016: ABA Fluffed Off Concerns About DDoS Attacks. Available from: <https://www.gizmodo.com.au/2017/03/censusfail-2016-abs-fluffed-off-concerns-about-ddos-attacks/> [Accessed: 10th October, 2017]
- Mahmood, Z. (2011), "Data Location and Security Issues in Cloud Computing" *International Conference on Emerging Intelligent Data and Web Technologies, IEEE*. Available from: <http://ieeexplore.ieee.org/document/6076420/> [Accessed: 10th October, 2016]
- Maiwald, E. (2003). *Network Security: A Beginner's Guide*. (2nd edn). McGraw-Hill/Osborne. New York. ISBN: 0072229578
- Mathematics Stack Exchange, (2011). Addition and Multiplication in a Galois Field Available at: <http://math.stackexchange.com/questions/89805/addition-and-multiplication-in-a-galois-field>
- McMillan, R., Knutson, R. (2017). Yahoo Triples Estimate of Breached Accounts to 3 Billion. [Online]. Available from: <https://www.wsj.com/articles/yahoo-triples-estimate-of-breachedaccounts-to-3-billion-1507062804> [Accessed: 11th Oct., 2017]
- Mell, P., Grance, T. (2011). *The NIST Definition of cloud computing*. [Online] Available from: <http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf> [Accessed: 1st Sept., 2015]
- Merkow, M. and Breithaupt, J. (2006). *Information Security Principles and Practices*. Pearson Prentice Hall. ISBN: 0131547291

- Natarajan, R. (2012). Apache Hadoop Fundamentals – HDFS and MapReduce Explained with a Diagram. [Online]. Available from: <http://www.thegeekstuff.com/2012/01/hadoop-hdfs-mapreduceintro/comment-page-1/> [Accessed: 15th June, 2014]
- O'Reilly, J. (2017). 7 Ways to Secure Cloud Storage. [Online] Available from: <https://www.networkcomputing.com/data-centers/7-ways-secure-cloud-storage/866645128> [Accessed: 15th Oct., 2017]
- OPC (2011). *Fact Sheet: Introduction to Cloud Computing*. [Online] Available from: https://www.priv.gc.ca/resource/fs-fi/02_05_d_51_cc_e.pdf [Accessed: 4th Sept., 2015]
- OpenCirrus (2017). *Cloud Computing Challenges In 2017*. [Online] Available from: <http://www.opencirrus.org/cloud-computing-challenges-2017/> [Accessed: 4th Sept., 2015]
- Perumal, S. and Kritzinger, P. (2004). *A tutorial on RAID storage systems*. [Online] Available from: http://23.pubs.cs.uct.ac.za/archive/00000131/01/perumal2004_RAIDTutorial.pdf [Accessed: 1st Feb., 2015]
- Plank, J. S. (2013). Erasure Codes for Storage Systems. Available from: https://www.usenix.org/system/files/login/articles/10_plank-online.pdf [Accessed: 10th May, 2015]
- Plank, J. S. (1997). A Tutorial on Reed-Solomon Coding for Fault-Tolerance in RAID-like Systems. *Software-Practice and Experience*. Vol.27, No.9, 995–1012. Available from: <http://cgi.di.uoa.gr/~ad/M155/Papers/RS-Tutorial.pdf> [Accessed: 10th May, 2015]
- Raisian, K. and Yahaya, J. (2015), "Security Issues Model on Cloud Computing: A Case of Malaysia", *International Journal of Advanced Computer Science and Applications*, Vol. 6, No. 8, pp.216-223.
- Rao, R. V. and Selvamani, K. (2015), "Data security challenges and its solutions in cloud computing", *Procedia Computer Science*, Vol. 48, pp. 204-209.
- REDTITAN, (2011). Error detection and correction. Available from: <http://www.pclviewer.com/rs2/galois.html> [Accessed: 10th May, 2015]
- Roshan, B. (2014). General Architecture of the Google File System. [Online]. Available from: <http://programming-project.blogspot.com/2014/04/general-architecture-of-google-file.html> [Accessed: 13th March, 2015]
- Rubens, P. (2017). Six Top CASB Vendors. [Online]. Available from: <https://www.esecurityplanet.com/products/top-casb-vendors.html> [Accessed: 5th Nov., 2017]
- Sailaja, K. and Usharani, M. (2017), "Cloud Computing Security Issues, Challenges and its Solutions in Financial Sectors", *International Journal of Advanced Scientific Technologies, Engineering and Managemnt*, Vol. 3, No.1, pp. 190-196.
- Satyanarayana, S. (2012), "Cloud Computing: SAAS", *GESJ: Computer Science and Telecommunications*, Vol. 36, No. 4, pp. 76-79
- Sebastian, A., Bonna, K. (Date). Reed-Solomon Encoder and Decoder. Available from:

<https://content.sakai.rutgers.edu/access/content/user/ak892/Reed-SolomonProjectReport.pdf>

[Accessed: 10th May, 2015]

Shah, H., Anandane, S. S., (2013), “Security Issues on Cloud Computing” *International Journal of Computer Science and Information Security*, Vol. 11, No. 8, pp. 25-33

Shapland, R. (2017). Multi-Tenancy Cloud Security Requires Enterprise Awareness. Available from: <http://searchcloudsecurity.techtarget.com/tip/Avoid-the-risks-of-multi-tenant-cloud-environmentsthrough-awareness> [Accessed: 10th October, 2017]

Shearer, D. (2017). Natural Disasters put the ‘A’ in the CIA Triad to test. Available from: http://blog.isc2.org/isc2_blog/2017/09/natural-disasters-put-the-a-in-the-cia-triad-to-test.html [Accessed: 10th October, 2017]

SkyHigh (2017). What is CASB? Available from: <https://www.skyhighnetworks.com/cloud-securityuniversity/what-is-cloud-access-security-broker/> [Accessed: 10th October, 2017]

Sommerville, I. (2001). Software Engineering. (9th edn). Addison-Wesley publications. ISBN-13: 978-0-13-703515-1

Stallings, W. (2011). Network Security Essentials: Applications and Standards. (4th edn). Pearson Education, Inc., Prentice Hall. Chapter 2. ISBN: 9780136108054

Stallings, W. (2003). Network Security Essentials: Applications and Standards. (2nd International edn). Upper Saddle River, NJ: Pearson Education. Chapter 2. ISBN: 0131202715

Strickland, J. (2017). How the Google File System Works. [Online]. Available from: <http://computer.howstuffworks.com/internet/basics/google-file-system5.htm> [Accessed: 15th March, 2017]

Tanenbaum, A. S. (2003). Computer Networks. (4th edn). Upper Saddle River, N.J.: Prentice Hall. Chapter 8, pgs. 724-750. ISBN: 0130384887

Tashi, J. and Ponsam, J. G. (2016), “Two tier Security Scheme for Storing and Retrieval of personal data in Cloud Storage”, *International Journal of Applied Engineering Research*, Vol. 11, No. 6, pp. 4354-4357

Tayseer, T. A. O., Amin, B. A. N. M, (2015). Internal and External Attacks in Cloud Computing Environment from Confidentiality, Integrity, and Availability points of view. *Journal of Computer Engineering*, Vol. 17, No. 2, pp. 93-96

Techopedia, (2017). Google File System (GFS). [Online]. Available from: <https://www.techopedia.com/definition/26906/google-file-system-gfs> [Accessed: 15th March, 2017]

The Treacherous 12, (2017). The Treacherous 12: Cloud Computing Top Threats in 2016 [Online]. Available from: <http://www.storm-clouds.eu/services/2017/04/the-treacherous-12-cloud-computingtop-threats-in-2016/> [Accessed: 25th October, 2017]

TipTopSecurity,(2016).Is Google Drive Safe to Use? How Google Secures Your Files Online

[Online]. Available from: <https://tiptopsecurity.com/is-google-drive-safe-to-use/> [Accessed: 1st Nov., 2017]

Trench W. F., (2003). *Introduction to Real Analysis*. Library of Congress Cataloging-in-Publication Data. Available from:

http://ramanujan.math.trinity.edu/wtrench/texts/TRENCH_REAL_ANALYSIS.PDF [Accessed: 10th May, 2015]

Trigueros-Preciado S., Perez-Gonzalez D., Solana-Gonzalez P., (2013). "Cloud computing in industrial SMEs: identification of the barriers to its adoption and effects of its application" *Electronic Mark*, Vol. 23, No. 2, pp. 105-114

Twum F, Hayfron-Acquah, J. B., Oblitey, W. W., Morgan-Darko, W., (2016a). Reed Solomon Encoding: Simplified explanation for Programmers. *International Journal of Computer Science and Information Security (IJCSIS)*, Vol. 14, No. 12

Twum F, Hayfron-Acquah, J. B., Oblitey, W. W., Boadi, R. K., (2016b). A proposed algorithm for generating the Reed-Solomon Encoding Polynomial Coefficients over GF(256) for RS[255,223]8,32. *International Journal of Computer Applications (IJCA)*, Vol. 156, No. 1, pgs. 24-39

Twum F, Hayfron-Acquah, J. B., Oblitey, W. W., Morgan-Darko, W., (2017). Reed Solomon Decoding Simplified for Programmers. *International Journal of Computer Science and Information Security (IJCSIS)*, Vol. 15, No. 1

Ukil A., Jana D., Sarkar A., (2013). A security framework in cloud computing infrastructure. *International Journal of Network Security and its Applications (IJNSA)*, Vol. 5, No.5

VMware (2009). *Securing the cloud*. [Online] Available from: <https://www.vmware.com/files/pdf/cloud/VMware-Savvis-Cloud-WP-en.pdf> [Accessed: 7th Sept., 2015]

Wall, M. (2017). Can we trust Cloud Providers to keep our data safe? [Online]. Available from: <http://www.bbc.com/news/business-36151754> [Accessed: 7th Sept., 2017]

Wang, J. (2009). *Computer Network Security Theory and Practice*. Springer
Wellenzohn, K. (2015). Erasure coding in distributed storage systems. Available from: http://webserver.inf.unibz.it/dis/teaching/SDB/reports/report_wellenzohn.pdf [Accessed: 7th Sept., 2015]

Wikiversity, (2016). Reed-Solomon Codes for Coders. Available from: https://en.wikiversity.org/wiki/Reed%E2%80%93Solomon_codes_for_coders [Accessed: 7th Sept., 2015]

Wei, W. (2016). Insider Breach: T-Mobile Czech Employee Steals and Sells 1.5 Million Users Data. Available from: <https://thehackernews.com/2016/06/t-mobile-hacked.html> [Accessed: 15th Oct., 2017]

Youssef, A. E., Alageel, M., (2012), "A Framework for Securing Cloud Computing", *International Journal of Computer Science Issues*, Vol. 9, No. 4, pp. 487-500.

KNUST



APPENDIX 1

Table A1 below presents the elements of GF(256)

Field Element	Alpha Exponent	Element Polynomial	Exponent Values		Log of Element
			Binary Representation	Decimal Representation	
0	(undefined)	0	00000000	0	(undefined)
1	α_0	α_0	00000001	1	0
2	α_1	α	00000010	2	1
3	α_2	α^2	00000100	4	25
4	α_3	α^3	00001000	8	2
5	α_4	α^4	00010000	16	50
6	α_5	α^5	00100000	32	26
7	α_6	α^6	01000000	64	198
8	α_7	α^7	10000000	128	3
9	α_8	$\alpha^4 + \alpha^3 + \alpha^2 + \alpha$	00011101	29	223
10	α_9	$\alpha^5 + \alpha^4 + \alpha^3 + \alpha$	00111010	58	51
11	α_{10}	$\alpha^6 + \alpha^5 + \alpha^4 + \alpha$	01110100	116	238
12	α_{11}	$\alpha^7 + \alpha^6 + \alpha^5 + \alpha$	11101000	232	27
13	α_{12}	$\alpha^7 + \alpha^6 + \alpha^3 + \alpha^2 + \alpha$	11001101	205	104
14	α_{13}	$\alpha^7 + \alpha^2 + \alpha + \alpha$	10000111	135	199
15	α_{14}	$\alpha^4 + \alpha + \alpha$	00010011	19	75
16	α_{15}	$\alpha^5 + \alpha + \alpha$	00100110	38	4
17	α_{16}	$\alpha^6 + \alpha + \alpha$	01001100	76	100
18	α_{17}	$\alpha^7 + \alpha + \alpha$	10011000	152	224

19	α_{18}	$\begin{matrix} 5 & 3 & 2 & 0 \\ \alpha + \alpha + \alpha + \alpha \end{matrix}$	00101101	45	14
20	α_{19}	$\begin{matrix} 6 & 4 & 3 & 1 \\ \alpha + \alpha + \alpha + \alpha \end{matrix}$	01011010	90	52
21	α_{20}	$\begin{matrix} 7 & 5 & 4 & 2 \\ \alpha + \alpha + \alpha + \alpha \end{matrix}$	10110100	180	141
22	α_{21}	$\begin{matrix} 6 & 5 & 4 & 2 & 0 \\ \alpha + \alpha + \alpha + \alpha + \alpha \end{matrix}$	01110101	117	239
23	α_{22}	$\begin{matrix} 7 & 6 & 5 & 3 & 1 \\ \alpha + \alpha + \alpha + \alpha + \alpha \end{matrix}$	11101010	234	129
24	α_{23}	$\begin{matrix} 7 & 6 & 3 & 0 \\ \alpha + \alpha + \alpha + \alpha \end{matrix}$	11001001	201	28
25	α_{24}	$\begin{matrix} 7 & 3 & 2 & 1 & 0 \\ \alpha + \alpha + \alpha + \alpha + \alpha \end{matrix}$	10001111	143	193
26	α_{25}	$\begin{matrix} 1 & 0 \\ \alpha + \alpha \end{matrix}$	00000011	3	105
27	α_{26}	$\begin{matrix} 2 & 1 \\ \alpha + \alpha \end{matrix}$	00000110	6	248

28	α_{27}	$\begin{matrix} 3 & 2 \\ \alpha + \alpha \end{matrix}$	00001100	12	200
29	α_{28}	$\begin{matrix} 4 & 3 \\ \alpha + \alpha \end{matrix}$	00011000	24	8
30	α_{29}	$\begin{matrix} 5 & 4 \\ \alpha + \alpha \end{matrix}$	00110000	48	76
31	α_{30}	$\begin{matrix} 6 & 5 \\ \alpha + \alpha \end{matrix}$	01100000	96	113
32	α_{31}	$\begin{matrix} 7 & 6 \\ \alpha + \alpha \end{matrix}$	11000000	192	5
33	α_{32}	$\begin{matrix} 7 & 4 & 3 & 2 & 0 \\ \alpha + \alpha + \alpha + \alpha + \alpha \end{matrix}$	10011101	157	138
34	α_{33}	$\begin{matrix} 5 & 2 & 1 & 0 \\ \alpha + \alpha + \alpha + \alpha \end{matrix}$	00100111	39	101
35	α_{34}	$\begin{matrix} 6 & 3 & 2 & 1 \\ \alpha + \alpha + \alpha + \alpha \end{matrix}$	01001110	78	47
36	α_{35}	$\begin{matrix} 7 & 4 & 3 & 2 \\ \alpha + \alpha + \alpha + \alpha \end{matrix}$	10011100	156	225
37	α_{36}	$\begin{matrix} 5 & 2 & 0 \\ \alpha + \alpha + \alpha \end{matrix}$	00100101	37	36
38	α_{37}	$\begin{matrix} 6 & 3 & 1 \\ \alpha + \alpha + \alpha \end{matrix}$	01001010	74	15
39	α_{38}	$\begin{matrix} 7 & 4 & 2 \\ \alpha + \alpha + \alpha \end{matrix}$	10010100	148	33
40	α_{39}	$\begin{matrix} 5 & 4 & 2 & 0 \\ \alpha + \alpha + \alpha + \alpha \end{matrix}$	00110101	53	53
41	α_{40}	$\begin{matrix} 6 & 5 & 3 & 1 \\ \alpha + \alpha + \alpha + \alpha \end{matrix}$	01101010	106	147

42	α_{41}	7 6 4 2 $\alpha + \alpha + \alpha + \alpha$	11010100	212	142
43	α_{42}	7 5 4 2 0 $\alpha + \alpha + \alpha + \alpha + \alpha$	10110101	181	218
44	α_{43}	6 5 4 2 1 0 $\alpha + \alpha + \alpha + \alpha + \alpha + \alpha$	01110111	119	240
45	α_{44}	7 6 5 3 2 1 $\alpha + \alpha + \alpha + \alpha + \alpha + \alpha$	11101110	238	18
46	α_{45}	7 6 0 $\alpha + \alpha + \alpha$	11000001	193	130
47	α_{46}	7 4 3 2 1 0 $\alpha + \alpha + \alpha + \alpha + \alpha + \alpha$	10011111	159	69
48	α_{47}	5 1 0 $\alpha + \alpha + \alpha$	00100011	35	29
49	α_{48}	6 2 1 $\alpha + \alpha + \alpha$	01000110	70	181
50	α_{49}	7 3 2 $\alpha + \alpha + \alpha$	10001100	140	194
51	α_{50}	2 0 $\alpha + \alpha$	00000101	5	125
52	α_{51}	3 1 $\alpha + \alpha$	00001010	10	106
53	α_{52}	4 2 $\alpha + \alpha$	00010100	20	39
54	α_{53}	5 3 $\alpha + \alpha$	00101000	40	249
55	α_{54}	6 4 $\alpha + \alpha$	01010000	80	185
56	α_{55}	7 5 $\alpha + \alpha$	10100000	160	201
57	α_{56}	6 4 3 2 0 $\alpha + \alpha + \alpha + \alpha + \alpha$	01011101	93	154
58	α_{57}	7 5 4 3 1 $\alpha + \alpha + \alpha + \alpha + \alpha$	10111010	186	9
59	α_{58}	6 5 3 0 $\alpha + \alpha + \alpha + \alpha$	01101001	105	120
60	α_{59}	7 6 4 1 $\alpha + \alpha + \alpha + \alpha$	11010010	210	77
61	α_{60}	7 5 4 3 0 $\alpha + \alpha + \alpha + \alpha + \alpha$	10111001	185	228

62	α_{61}	6 5 3 2 1 0 $\alpha + \alpha + \alpha + \alpha + \alpha + \alpha$	01101111	111	114
63	α_{62}	7 6 4 3 2 1 $\alpha + \alpha + \alpha + \alpha + \alpha + \alpha$	11011110	222	166
64	α_{63}	7 5 0 $\alpha + \alpha + \alpha$	10100001	161	6
65	α_{64}	6 4 3 2 1 0 $\alpha + \alpha + \alpha + \alpha + \alpha + \alpha$	01011111	95	191

66	α_{65}	7 5 4 3 2 1 $\alpha + \alpha + \alpha + \alpha + \alpha + \alpha$	10111110	190	139
67	α_{66}	6 5 0 $\alpha + \alpha + \alpha$	01100001	97	98
68	α_{67}	7 6 1 $\alpha + \alpha + \alpha$	11000010	194	102
69	α_{68}	7 4 3 0 $\alpha + \alpha + \alpha + \alpha$	10011001	153	221
70	α_{69}	5 3 2 1 0 $\alpha + \alpha + \alpha + \alpha + \alpha$	00101111	47	48
71	α_{70}	6 4 3 2 1 $\alpha + \alpha + \alpha + \alpha + \alpha$	01011110	94	253
72	α_{71}	7 5 4 3 2 $\alpha + \alpha + \alpha + \alpha + \alpha$	10111100	188	226
73	α_{72}	6 5 2 0 $\alpha + \alpha + \alpha + \alpha$	01100101	101	152
74	α_{73}	7 6 3 1 $\alpha + \alpha + \alpha + \alpha$	11001010	202	37
75	α_{74}	7 3 0 $\alpha + \alpha + \alpha$	10001001	137	179
76	α_{75}	3 2 1 0 $\alpha + \alpha + \alpha + \alpha$	00001111	15	16
77	α_{76}	4 3 2 1 $\alpha + \alpha + \alpha + \alpha$	00011110	30	145
78	α_{77}	5 4 3 2 $\alpha + \alpha + \alpha + \alpha$	00111100	60	34
79	α_{78}	6 5 4 3 $\alpha + \alpha + \alpha + \alpha$	01111000	120	136
80	α_{79}	7 6 5 4 $\alpha + \alpha + \alpha + \alpha$	11110000	240	54
81	α_{80}	7 6 5 4 3 2 0 $\alpha + \alpha + \alpha + \alpha + \alpha + \alpha + \alpha$	11111101	253	208
82	α_{81}	7 6 5 2 1 0 $\alpha + \alpha + \alpha + \alpha + \alpha + \alpha$	11100111	231	148
83	α_{82}	7 6 4 1 0 $\alpha + \alpha + \alpha + \alpha + \alpha$	11010011	211	206
84	α_{83}	7 5 4 3 1 0 $\alpha + \alpha + \alpha + \alpha + \alpha + \alpha$	10111011	187	143
85	α_{84}	6 5 3 1 0 $\alpha + \alpha + \alpha + \alpha + \alpha$	01101011	107	150
86	α_{85}	7 6 4 2 1 $\alpha + \alpha + \alpha + \alpha + \alpha$	11010110	214	219
87	α_{86}	7 5 4 0 $\alpha + \alpha + \alpha + \alpha$	10110001	177	189
88	α_{87}	6 5 4 3 2 1 0 $\alpha + \alpha + \alpha + \alpha + \alpha + \alpha + \alpha$	01111111	127	241
89	α_{88}	7 6 5 4 3 2 1 $\alpha + \alpha + \alpha + \alpha + \alpha + \alpha + \alpha$	11111110	254	210

90	α_{89}	7 6 5 0 $\alpha + \alpha + \alpha + \alpha$	11100001	225	19
91	α_{90}	7 6 4 3 2 1 0 $\alpha + \alpha + \alpha + \alpha + \alpha + \alpha + \alpha$	11011111	223	92
92	α_{91}	7 5 1 0 $\alpha + \alpha + \alpha + \alpha$	10100011	163	131
93	α_{92}	6 4 3 1 0 $\alpha + \alpha + \alpha + \alpha + \alpha$	01011011	91	56
94	α_{93}	7 5 4 2 1 $\alpha + \alpha + \alpha + \alpha + \alpha$	10110110	182	70
95	α_{94}	6 5 4 0 $\alpha + \alpha + \alpha + \alpha$	01110001	113	64
96	α_{95}	7 6 5 1 $\alpha + \alpha + \alpha + \alpha$	11100010	226	30
97	α_{96}	7 6 4 3 0 $\alpha + \alpha + \alpha + \alpha + \alpha$	11011001	217	66
98	α_{97}	7 5 3 2 1 0 $\alpha + \alpha + \alpha + \alpha + \alpha + \alpha$	10101111	175	182
99	α_{98}	6 1 0 $\alpha + \alpha + \alpha$	01000011	67	163
100	α_{99}	7 2 1 $\alpha + \alpha + \alpha$	10000110	134	195
101	α_{100}	4 0 $\alpha + \alpha$	00010001	17	72
102	α_{101}	5 1 $\alpha + \alpha$	00100010	34	126
103	α_{102}	6 2 $\alpha + \alpha$	01000100	68	110
104	α_{103}	7 3 $\alpha + \alpha$	10001000	136	107
105	α_{104}	3 2 0 $\alpha + \alpha + \alpha$	00001101	13	58
106	α_{105}	4 3 1 $\alpha + \alpha + \alpha$	00011010	26	40
107	α_{106}	5 4 2 $\alpha + \alpha + \alpha$	00110100	52	84
108	α_{107}	6 5 3 $\alpha + \alpha + \alpha$	01101000	104	250
109	α_{108}	7 6 4 $\alpha + \alpha + \alpha$	11010000	208	133
110	α_{109}	7 5 4 3 2 0 $\alpha + \alpha + \alpha + \alpha + \alpha + \alpha$	10111101	189	186
111	α_{110}	6 5 2 1 0 $\alpha + \alpha + \alpha + \alpha + \alpha$	01100111	103	61
112	α_{111}	7 6 3 2 1 $\alpha + \alpha + \alpha + \alpha + \alpha$	11001110	206	202
113	α_{112}	7 0 $\alpha + \alpha$	10000001	129	94

114	α_{113}	4 3 2 1 0 $\alpha + \alpha + \alpha + \alpha + \alpha$	00011111	31	155
115	α_{114}	5 4 3 2 1 $\alpha + \alpha + \alpha + \alpha + \alpha$	00111110	62	159
116	α_{115}	6 5 4 3 2 $\alpha + \alpha + \alpha + \alpha + \alpha$	01111100	124	10
117	α_{116}	7 6 5 4 3 $\alpha + \alpha + \alpha + \alpha + \alpha$	11111000	248	21
118	α_{117}	7 6 5 3 2 0 $\alpha + \alpha + \alpha + \alpha + \alpha + \alpha$	11101101	237	121
119	α_{118}	7 6 2 1 0 $\alpha + \alpha + \alpha + \alpha + \alpha$	11000111	199	43
120	α_{119}	7 4 1 0 $\alpha + \alpha + \alpha + \alpha$	10010011	147	78
121	α_{120}	5 4 3 1 0 $\alpha + \alpha + \alpha + \alpha + \alpha$	00111011	59	212
122	α_{121}	6 5 4 2 1 $\alpha + \alpha + \alpha + \alpha + \alpha$	01110110	118	229
123	α_{122}	7 6 5 3 2 $\alpha + \alpha + \alpha + \alpha + \alpha$	11101100	236	172
124	α_{123}	7 6 2 0 $\alpha + \alpha + \alpha + \alpha$	11000101	197	115
125	α_{124}	7 4 2 1 0 $\alpha + \alpha + \alpha + \alpha + \alpha$	10010111	151	243

126	α_{125}	5 4 1 0 $\alpha + \alpha + \alpha + \alpha$	00110011	51	167
127	α_{126}	6 5 2 1 $\alpha + \alpha + \alpha + \alpha$	01100110	102	87
128	α_{127}	7 6 3 2 $\alpha + \alpha + \alpha + \alpha$	11001100	204	7
129	α_{128}	7 2 0 $\alpha + \alpha + \alpha$	10000101	133	112
130	α_{129}	4 2 1 0 $\alpha + \alpha + \alpha + \alpha$	00010111	23	192
131	α_{130}	5 3 2 1 $\alpha + \alpha + \alpha + \alpha$	00101110	46	247
132	α_{131}	6 4 3 2 $\alpha + \alpha + \alpha + \alpha$	01011100	92	140
133	α_{132}	7 5 4 3 $\alpha + \alpha + \alpha + \alpha$	10111000	184	128
134	α_{133}	6 5 3 2 0 $\alpha + \alpha + \alpha + \alpha + \alpha$	01101101	109	99
135	α_{134}	7 6 4 3 1 $\alpha + \alpha + \alpha + \alpha + \alpha$	11011010	218	13
136	α_{135}	7 5 3 0 $\alpha + \alpha + \alpha + \alpha$	10101001	169	103
137	α_{136}	6 3 2 1 0 $\alpha + \alpha + \alpha + \alpha + \alpha$	01001111	79	74

138	α_{137}	7 4 3 2 1 $\alpha + \alpha + \alpha + \alpha + \alpha$	10011110	158	222
139	α_{138}	5 0 $\alpha + \alpha$	00100001	33	237
140	α_{139}	6 1 $\alpha + \alpha$	01000010	66	49
141	α_{140}	7 2 $\alpha + \alpha$	10000100	132	197
142	α_{141}	4 2 0 $\alpha + \alpha + \alpha$	00010101	21	254
143	α_{142}	5 3 1 $\alpha + \alpha + \alpha$	00101010	42	24
144	α_{143}	6 4 2 $\alpha + \alpha + \alpha$	01010100	84	227
145	α_{144}	7 5 3 $\alpha + \alpha + \alpha$	10101000	168	165
146	α_{145}	6 3 2 0 $\alpha + \alpha + \alpha + \alpha$	01001101	77	153
147	α_{146}	7 4 3 1 $\alpha + \alpha + \alpha + \alpha$	10011010	154	119
148	α_{147}	5 3 0 $\alpha + \alpha + \alpha$	00101001	41	38
149	α_{148}	6 4 1 $\alpha + \alpha + \alpha$	01010010	82	184
150	α_{149}	7 5 2 $\alpha + \alpha + \alpha$	10100100	164	180
151	α_{150}	6 4 2 0 $\alpha + \alpha + \alpha + \alpha$	01010101	85	124
152	α_{151}	7 5 3 1 $\alpha + \alpha + \alpha + \alpha$	10101010	170	17
153	α_{152}	6 3 0 $\alpha + \alpha + \alpha$	01001001	73	68
154	α_{153}	7 4 1 $\alpha + \alpha + \alpha$	10010010	146	146
155	α_{154}	5 4 3 0 $\alpha + \alpha + \alpha + \alpha$	00111001	57	217
156	α_{155}	6 5 4 1 $\alpha + \alpha + \alpha + \alpha$	01110010	114	35
157	α_{156}	7 6 5 2 $\alpha + \alpha + \alpha + \alpha$	11100100	228	32
158	α_{157}	7 6 4 2 0 $\alpha + \alpha + \alpha + \alpha + \alpha$	11010101	213	137
159	α_{158}	7 5 4 2 1 0 $\alpha + \alpha + \alpha + \alpha + \alpha + \alpha$	10110111	183	46
160	α_{159}	6 5 4 1 0 $\alpha + \alpha + \alpha + \alpha + \alpha$	01110011	115	55
161	α_{160}	7 6 5 2 1 $\alpha + \alpha + \alpha + \alpha + \alpha$	11100110	230	63

162	α_{161}	7 6 4 0 $\alpha + \alpha + \alpha + \alpha$	11010001	209	209
163	α_{162}	7 5 4 3 2 1 0 $\alpha + \alpha + \alpha + \alpha + \alpha + \alpha + \alpha$	10111111	191	91
164	α_{163}	6 5 1 0 $\alpha + \alpha + \alpha + \alpha$	01100011	99	149
165	α_{164}	7 6 2 1 $\alpha + \alpha + \alpha + \alpha$	11000110	198	188
166	α_{165}	7 4 0 $\alpha + \alpha + \alpha$	10010001	145	207
167	α_{166}	5 4 3 2 1 0 $\alpha + \alpha + \alpha + \alpha + \alpha + \alpha$	00111111	63	205
168	α_{167}	6 5 4 3 2 1 $\alpha + \alpha + \alpha + \alpha + \alpha + \alpha$	01111110	126	144
169	α_{168}	7 6 5 4 3 2 $\alpha + \alpha + \alpha + \alpha + \alpha + \alpha$	11111100	252	135
170	α_{169}	7 6 5 2 0 $\alpha + \alpha + \alpha + \alpha + \alpha$	11100101	229	151
171	α_{170}	7 6 4 2 1 0 $\alpha + \alpha + \alpha + \alpha + \alpha + \alpha$	11010111	215	178
172	α_{171}	7 5 4 1 0 $\alpha + \alpha + \alpha + \alpha + \alpha$	10110011	179	220
173	α_{172}	6 5 4 3 1 0 $\alpha + \alpha + \alpha + \alpha + \alpha + \alpha$	01111011	123	252
174	α_{173}	7 6 5 4 2 1 $\alpha + \alpha + \alpha + \alpha + \alpha + \alpha$	11110110	246	190
175	α_{174}	7 6 5 4 0 $\alpha + \alpha + \alpha + \alpha + \alpha$	11110001	241	97
176	α_{175}	7 6 5 4 3 2 1 0 $\alpha + \alpha + \alpha + \alpha + \alpha + \alpha + \alpha + \alpha$	11111111	255	242
177	α_{176}	7 6 5 1 0 $\alpha + \alpha + \alpha + \alpha + \alpha$	11100011	227	86
178	α_{177}	7 6 4 3 1 0 $\alpha + \alpha + \alpha + \alpha + \alpha + \alpha$	11011011	219	211
179	α_{178}	7 5 3 1 0 $\alpha + \alpha + \alpha + \alpha + \alpha$	10101011	171	171
180	α_{179}	6 3 1 0 $\alpha + \alpha + \alpha + \alpha$	01001011	75	20
181	α_{180}	7 4 2 1 $\alpha + \alpha + \alpha + \alpha$	10010110	150	42
182	α_{181}	5 4 0 $\alpha + \alpha + \alpha$	00110001	49	93
183	α_{182}	6 5 1 $\alpha + \alpha + \alpha$	01100010	98	158
184	α_{183}	7 6 2 $\alpha + \alpha + \alpha$	11000100	196	132
185	α_{184}	7 4 2 0 $\alpha + \alpha + \alpha + \alpha$	10010101	149	60
186	α_{185}	5 4 2 1 0 $\alpha + \alpha + \alpha + \alpha + \alpha$	00110111	55	57

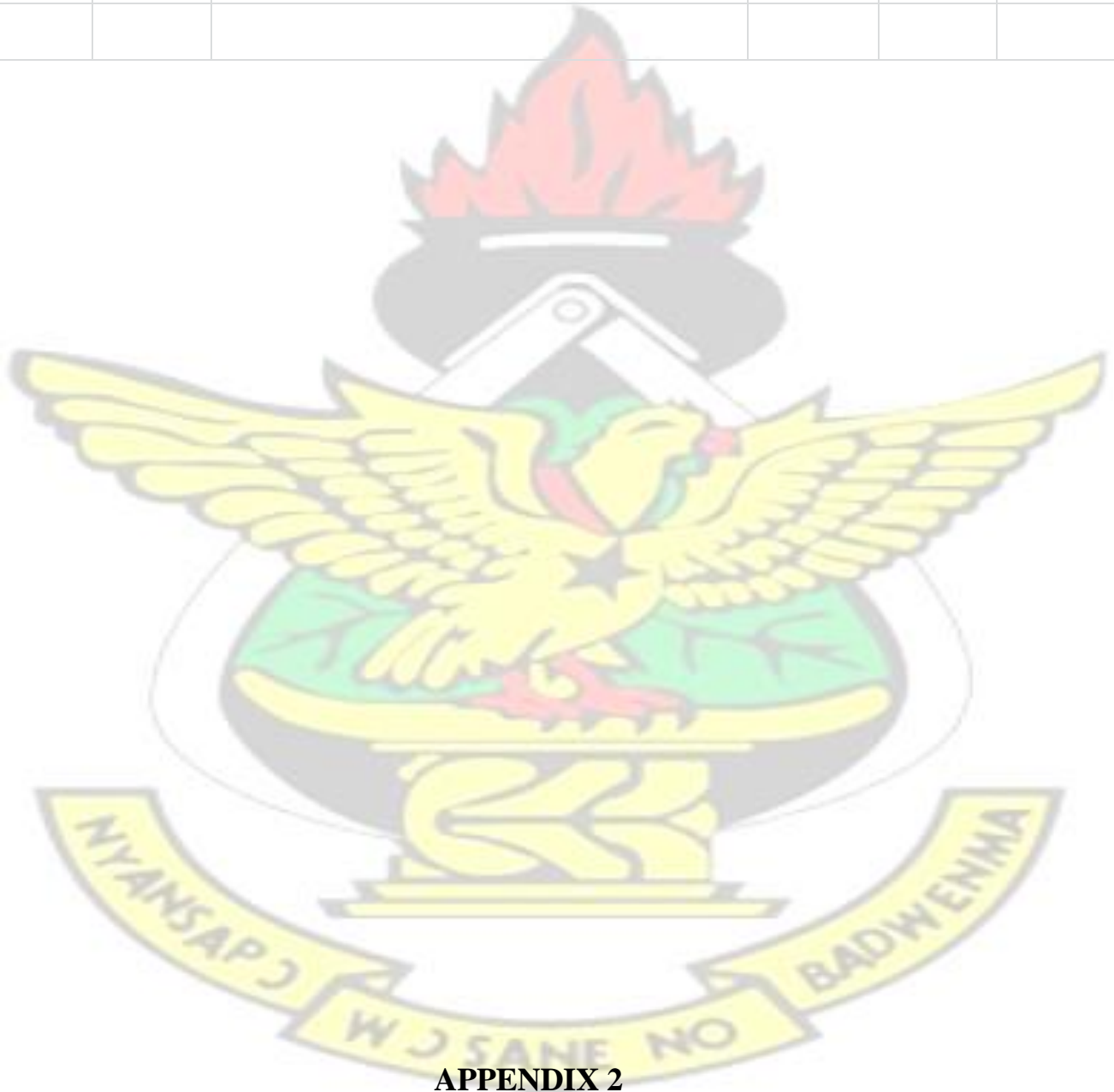
187	α_{186}	6 5 3 2 1 $\alpha + \alpha + \alpha + \alpha + \alpha$	01101110	110	83
188	α_{187}	7 6 4 3 2 $\alpha + \alpha + \alpha + \alpha + \alpha$	11011100	220	71
189	α_{188}	7 5 2 0 $\alpha + \alpha + \alpha + \alpha$	10100101	165	109
190	α_{189}	6 4 2 1 0 $\alpha + \alpha + \alpha + \alpha + \alpha$	01010111	87	65
191	α_{190}	7 5 3 2 1 $\alpha + \alpha + \alpha + \alpha + \alpha$	10101110	174	162
192	α_{191}	6 0 $\alpha + \alpha$	01000001	65	31
193	α_{192}	7 1 $\alpha + \alpha$	10000010	130	45
194	α_{193}	4 3 0 $\alpha + \alpha + \alpha$	00011001	25	67
195	α_{194}	5 4 1 $\alpha + \alpha + \alpha$	00110010	50	216
196	α_{195}	6 5 2 $\alpha + \alpha + \alpha$	01100100	100	183
197	α_{196}	7 6 3 $\alpha + \alpha + \alpha$	11001000	200	123
198	α_{197}	7 3 2 0 $\alpha + \alpha + \alpha + \alpha$	10001101	141	164
199	α_{198}	2 1 0 $\alpha + \alpha + \alpha$	00000111	7	118
200	α_{199}	3 2 1 $\alpha + \alpha + \alpha$	00001110	14	196
201	α_{200}	4 3 2 $\alpha + \alpha + \alpha$	00011100	28	23
202	α_{201}	5 4 3 $\alpha + \alpha + \alpha$	00111000	56	73
203	α_{202}	6 5 4 $\alpha + \alpha + \alpha$	01110000	112	236
204	α_{203}	7 6 5 $\alpha + \alpha + \alpha$	11100000	224	127
205	α_{204}	7 6 4 3 2 0 $\alpha + \alpha + \alpha + \alpha + \alpha + \alpha$	11011101	221	12
206	α_{205}	7 5 2 1 0 $\alpha + \alpha + \alpha + \alpha + \alpha$	10100111	167	111
207	α_{206}	6 4 1 0 $\alpha + \alpha + \alpha + \alpha$	01010011	83	246
208	α_{207}	7 5 2 1 $\alpha + \alpha + \alpha + \alpha$	10100110	166	108
209	α_{208}	6 4 0 $\alpha + \alpha + \alpha$	01010001	81	161

210	α_{209}	7 5 1 $\alpha + \alpha + \alpha$	10100010	162	59
211	α_{210}	6 4 3 0 $\alpha + \alpha + \alpha + \alpha$	01011001	89	82
212	α_{211}	7 5 4 1 $\alpha + \alpha + \alpha + \alpha$	10110010	178	41
213	α_{212}	6 5 4 3 0 $\alpha + \alpha + \alpha + \alpha + \alpha$	01111001	121	157
214	α_{213}	7 6 5 4 1 $\alpha + \alpha + \alpha + \alpha + \alpha$	11110010	242	85
215	α_{214}	7 6 5 4 3 0 $\alpha + \alpha + \alpha + \alpha + \alpha + \alpha$	11111001	249	170
216	α_{215}	7 6 5 3 2 1 0 $\alpha + \alpha + \alpha + \alpha + \alpha + \alpha + \alpha$	11101111	239	251
217	α_{216}	7 6 1 0 $\alpha + \alpha + \alpha + \alpha$	11000011	195	96
218	α_{217}	7 4 3 1 0 $\alpha + \alpha + \alpha + \alpha + \alpha$	10011011	155	134
219	α_{218}	5 3 1 0 $\alpha + \alpha + \alpha + \alpha$	00101011	43	177
220	α_{219}	6 4 2 1 $\alpha + \alpha + \alpha + \alpha$	01010110	86	187
221	α_{220}	7 5 3 2 $\alpha + \alpha + \alpha + \alpha$	10101100	172	204
222	α_{221}	6 2 0 $\alpha + \alpha + \alpha$	01000101	69	62
223	α_{222}	7 3 1 $\alpha + \alpha + \alpha$	10001010	138	90
224	α_{223}	3 0 $\alpha + \alpha$	00001001	9	203
225	α_{224}	4 1 $\alpha + \alpha$	00010010	18	89
226	α_{225}	5 2 $\alpha + \alpha$	00100100	36	95
227	α_{226}	6 3 $\alpha + \alpha$	01001000	72	176
228	α_{227}	7 4 $\alpha + \alpha$	10010000	144	156
229	α_{228}	5 4 3 2 0 $\alpha + \alpha + \alpha + \alpha + \alpha$	00111101	61	169
230	α_{229}	6 5 4 3 1 $\alpha + \alpha + \alpha + \alpha + \alpha$	01111010	122	160
231	α_{230}	7 6 5 4 2 $\alpha + \alpha + \alpha + \alpha + \alpha$	11110100	244	81
232	α_{231}	7 6 5 4 2 0 $\alpha + \alpha + \alpha + \alpha + \alpha + \alpha$	11110101	245	11

233	α_{232}	7 6 5 4 2 1 0 $\alpha + \alpha + \alpha + \alpha + \alpha + \alpha + \alpha$	11110111	247	245
234	α_{233}	7 6 5 4 1 0 $\alpha + \alpha + \alpha + \alpha + \alpha + \alpha$	11110011	243	22
235	α_{234}	7 6 5 4 3 1 0 $\alpha + \alpha + \alpha + \alpha + \alpha + \alpha + \alpha$	11111011	251	235
236	α_{235}	7 6 5 3 1 0 $\alpha + \alpha + \alpha + \alpha + \alpha + \alpha$	11101011	235	122
237	α_{236}	7 6 3 1 0 $\alpha + \alpha + \alpha + \alpha + \alpha$	11001011	203	117
238	α_{237}	7 3 1 0 $\alpha + \alpha + \alpha + \alpha$	10001011	139	44
239	α_{238}	3 1 0 $\alpha + \alpha + \alpha$	00001011	11	215
240	α_{239}	4 2 1 $\alpha + \alpha + \alpha$	00010110	22	79
241	α_{240}	5 3 2 $\alpha + \alpha + \alpha$	00101100	44	174
242	α_{241}	6 4 3 $\alpha + \alpha + \alpha$	01011000	88	213
243	α_{242}	7 5 4 $\alpha + \alpha + \alpha$	10110000	176	233
244	α_{243}	6 5 4 3 2 0 $\alpha + \alpha + \alpha + \alpha + \alpha + \alpha$	01111101	125	230
245	α_{244}	7 6 5 4 3 1 $\alpha + \alpha + \alpha + \alpha + \alpha + \alpha$	11111010	250	231
246	α_{245}	7 6 5 3 0 $\alpha + \alpha + \alpha + \alpha + \alpha$	11101001	233	173
247	α_{246}	7 6 3 2 1 0 $\alpha + \alpha + \alpha + \alpha + \alpha + \alpha$	11001111	207	232
248	α_{247}	7 1 0 $\alpha + \alpha + \alpha$	10000011	131	116
249	α_{248}	4 3 1 0 $\alpha + \alpha + \alpha + \alpha$	00011011	27	214

250	α_{249}	5 4 2 1 $\alpha + \alpha + \alpha + \alpha$	00110110	54	244
251	α_{250}	6 5 3 2 $\alpha + \alpha + \alpha + \alpha$	01101100	108	234
252	α_{251}	7 6 4 3 $\alpha + \alpha + \alpha + \alpha$	11011000	216	168

253	α_{252}	$\begin{matrix} & 7 & 5 & 3 & 2 & 0 \\ & \alpha + \alpha + \alpha + \alpha + \alpha \end{matrix}$	10101101	173	80
254	α_{253}	$\begin{matrix} & 6 & 2 & 1 & 0 \\ & \alpha + \alpha + \alpha + \alpha \end{matrix}$	01000111	71	88
255	α_{254}	$\begin{matrix} & 7 & 3 & 2 & 1 \\ & \alpha + \alpha + \alpha + \alpha \end{matrix}$	10001110	142	175
	α_{255}	α_0	00000001	1	



Proposed 'Secure My Files' System

Application Usage

Initial Steps

To make use of the application's upload and download functions, the system user has to perform a number of operations to prepare the application for interacting with the cloud. The initial activities are

Launch the application software:

The *Secure My Files* application provides an executable java archive which the user can launch by double-clicking or clicking once then pressing the “return” or “enter” key on the keyboard.

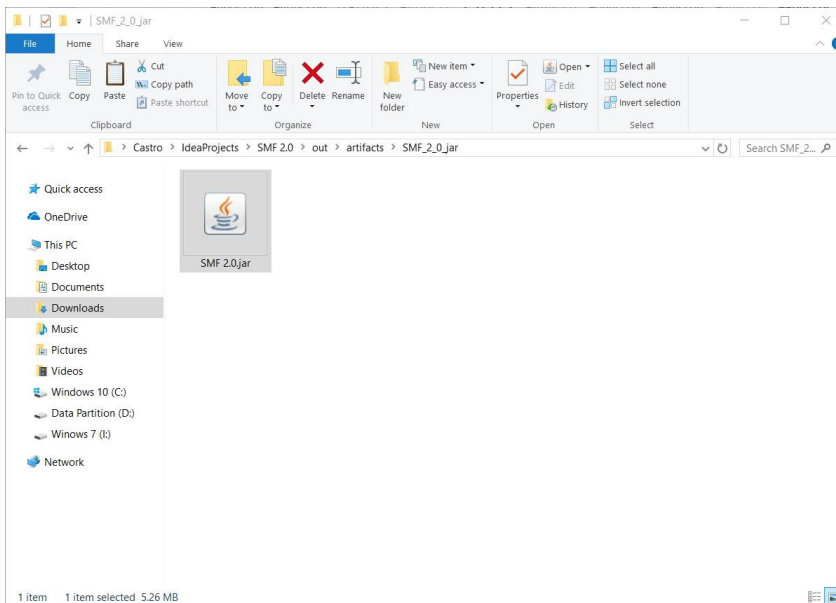


Figure 1: Executable java archive for Secure My Files, named “SMF 2.0.jar”

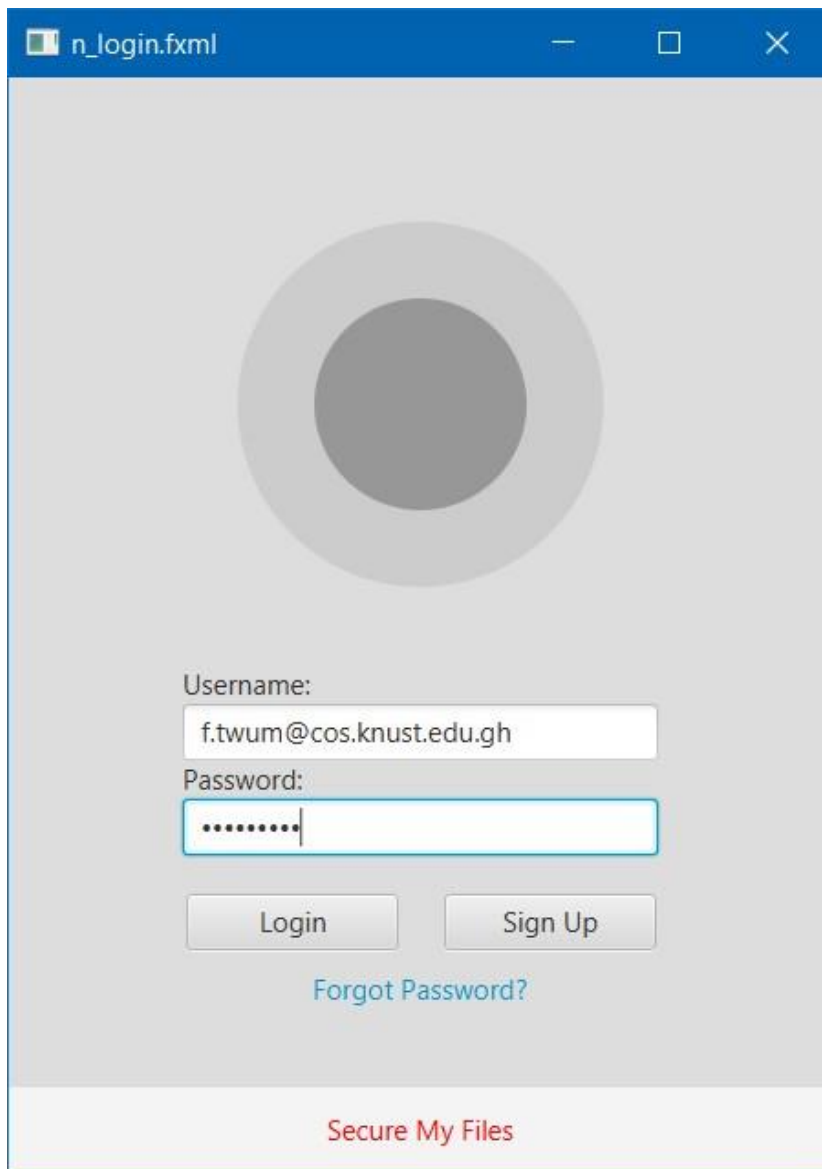
Log into the user's Secure My Files account

Upon startup, the user is presented with a login page to enter the following details:

Username: This is the username with which the user registered for the service, i.e. Secure My Files.

Password: This is the password which the user used when registering for Secure My Files.

After entering the details, the user clicks the “Login” button to access the main page of the application.



Username:

f.twum@cos.knust.edu.gh

Password:

.....

Login Sign Up

[Forgot Password?](#)

[Secure My Files](#)

Figure 2: Login page

If the user is not registered, the user may click the “Sign Up” button to open the registration page. On the registration page, the user is provided with input fields to enter the following information:

Full Name: This is the full name of the system user.

Secure My Files Username: This is the username (email address) that will be used to log into the service at a later period.

Password: This is the password that will be used to log into the Secure My Files application at a later period.

Confirm Password: This field asks the user to re-enter the password to ensure that the user is familiar with what was typed in the “Password” field.

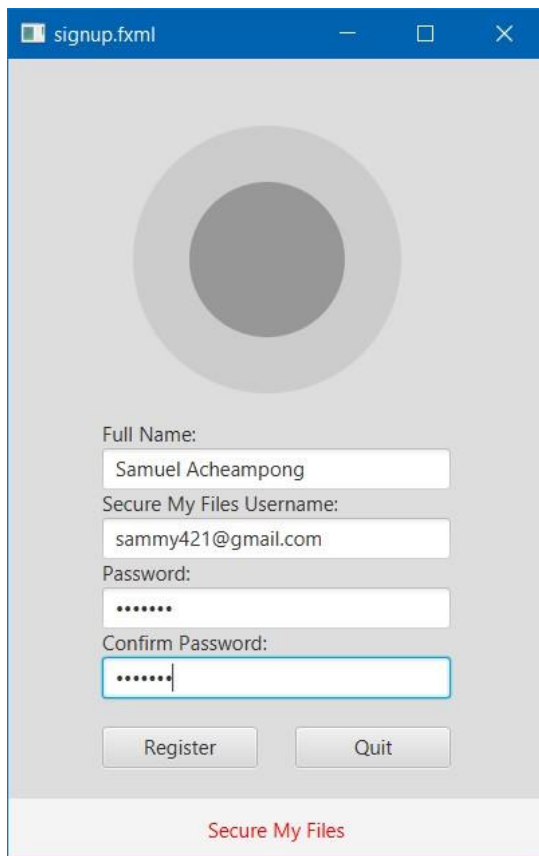
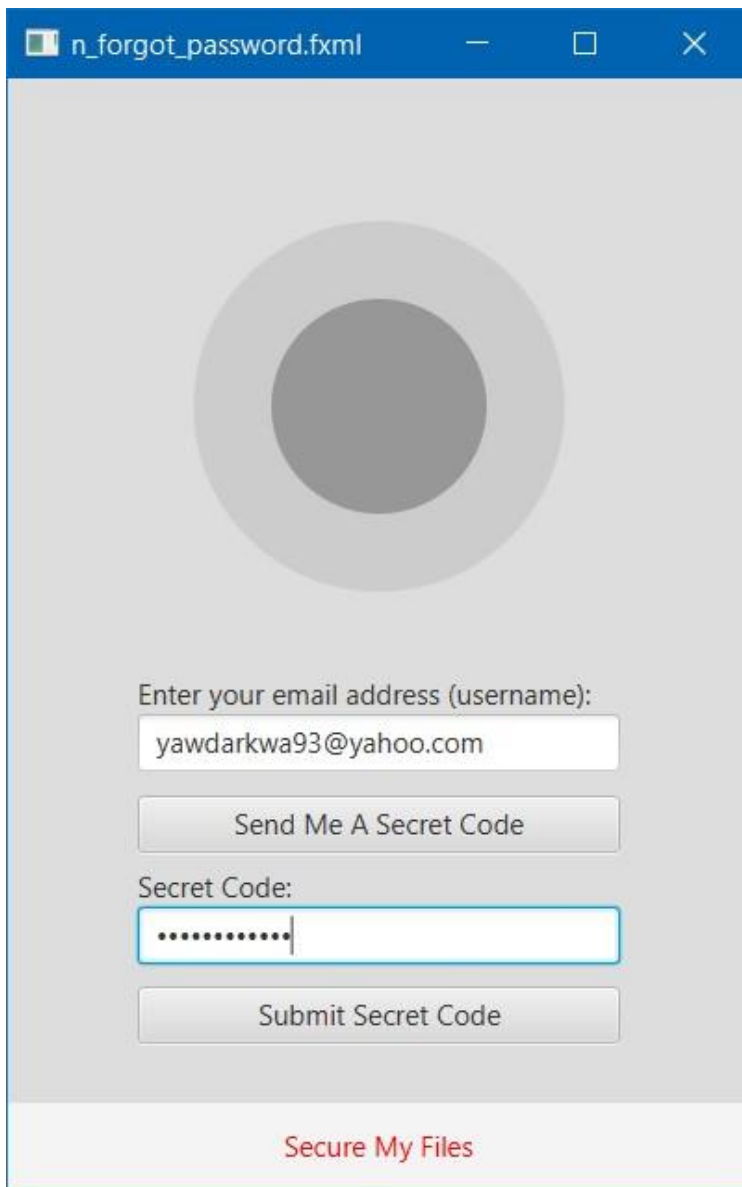


Figure 3: Registration page

Upon completing the registration form, the user then clicks the “Register” button to register for the service. The user may also select the “Quit” button to exit the application.

If the user is already registered but has forgotten the password associated with the account, the user may click the “Forgot Password?” hyperlink on the login page to initiate the “Password Recovery” process. The user is presented with the “Forgot Password” page that asks for the username associated with the user’s account. An email is sent to the user’s account with a secret code with which the user can change the associated password. After retrieving the secret code from the email sent by the application, the user enters the secret code into the field supplied in the “Password Recovery” page for that purpose.



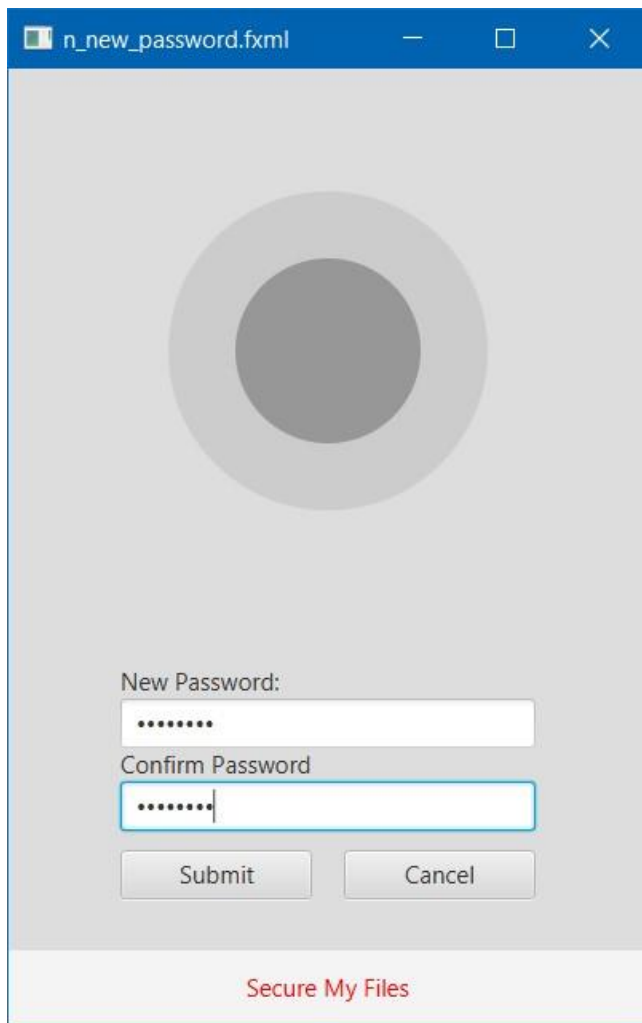
The screenshot shows a web browser window with the title "n_forgot_password.fxml". The page has a light gray background with a large, faint circular logo in the center. Below the logo, there is a form with the following elements:

- A label "Enter your email address (username):" followed by a text input field containing "yawdarkwa93@yahoo.com".
- A button labeled "Send Me A Secret Code".
- A label "Secret Code:" followed by a text input field containing a series of dots.
- A button labeled "Submit Secret Code".
- A red link labeled "Secure My Files" at the bottom.

Figure 4: Password Recovery initial page

After selecting the “Submit Secret Code” button, the application verifies the secret code and presents the user with the “New Password” page to enter and confirm a new password.

On the “New Password” page, the user is provided with a field to input a new password and another field to confirm the password. On entering the requisite data, the user may proceed to click the “Submit” button to complete the Password Change, or click the “Cancel” button to cancel the process.



n_new_password.fxml

— □ ×

NUST

New Password:

.....

Confirm Password

.....

Submit Cancel

Secure My Files

Figure 5: New Password page

Main Functionality

The main page of Secure My Files provides access to its functions by means of buttons and a menu bar. All the functions performed by the buttons can also be found in the menu bar.

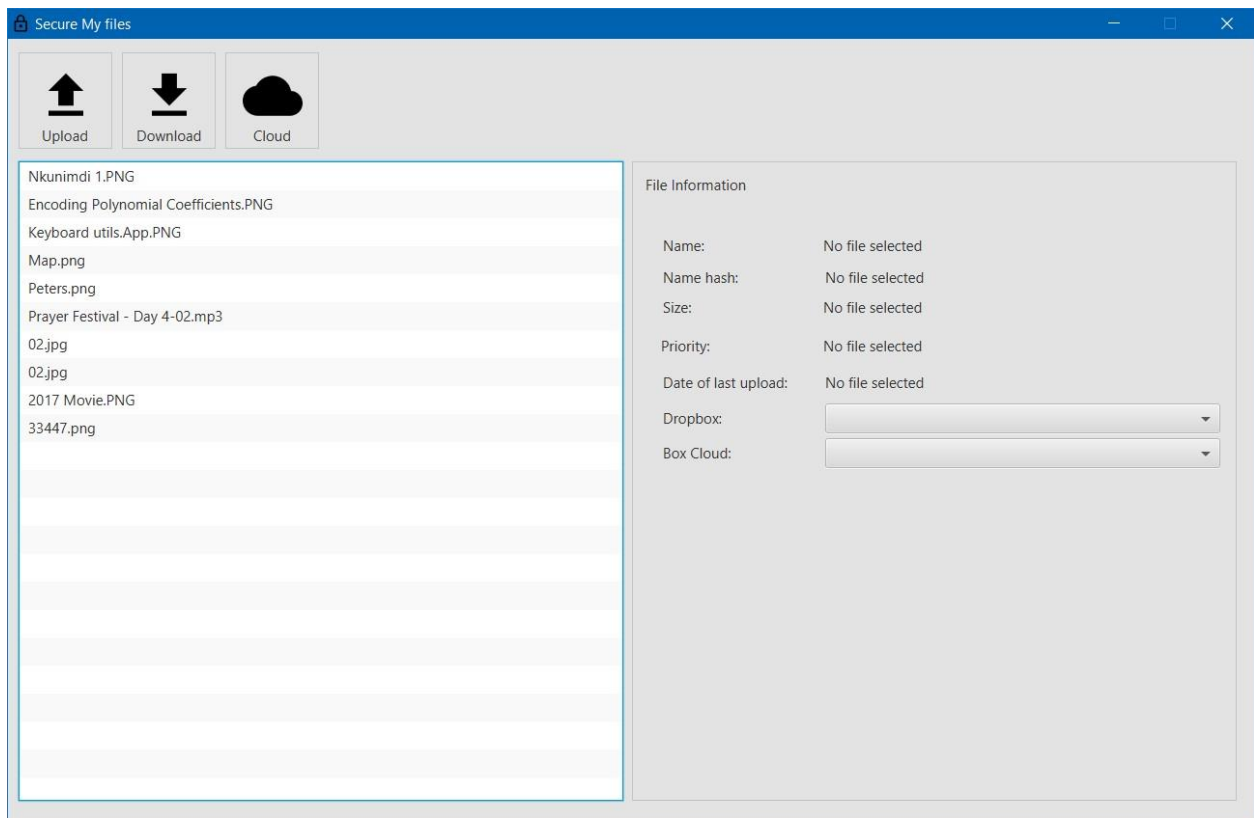


Figure 6: Secure My Files main page

The functions available to the user of the system are:

Upload a file

The system user can initiate a file upload by simply clicking on the upload button.



Figure 7: Upload button

The user is then presented with the “New Upload” page with options to:

Select a file

The user first clicks the “Select a file” button which opens a “Select file” dialogue from which the user can navigate to the desired file and choose it.

Specify the priority of the file

The user selects one of four radio buttons, for “Low”, “Normal”, “Important” and “Critical” priorities.

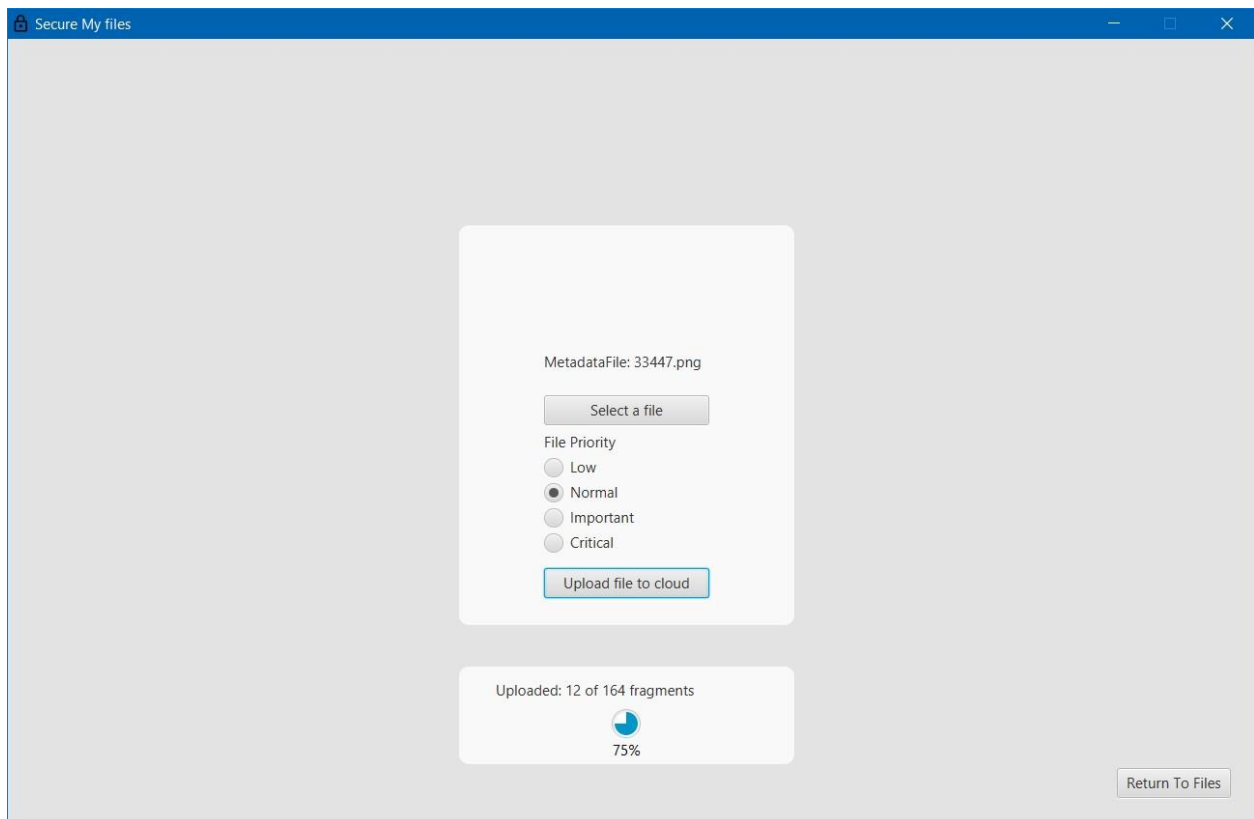


Figure 8: File Upload page showing the upload of a file named “33447.png”

The user proceeds to click the “Upload file to cloud” button to trigger the processes which hash the file name, encrypt the file, split the file into multiple shards, scramble the order of the file shards and finally upload the file to the user’s cloud accounts.

It should be noted that the user must first log into at least one cloud account for the upload to begin. If the user hasn’t yet signed into any cloud account, the system reacts to the upload button’s click event by displaying the cloud login page.

Sign into cloud accounts

System users must sign into their cloud accounts before an upload or download can be performed. To initiate signing into cloud accounts, the user simply clicks the “Cloud” button. The system then displays the “Cloud” page for logging into and out of the cloud services.



Figure 9: Cloud button

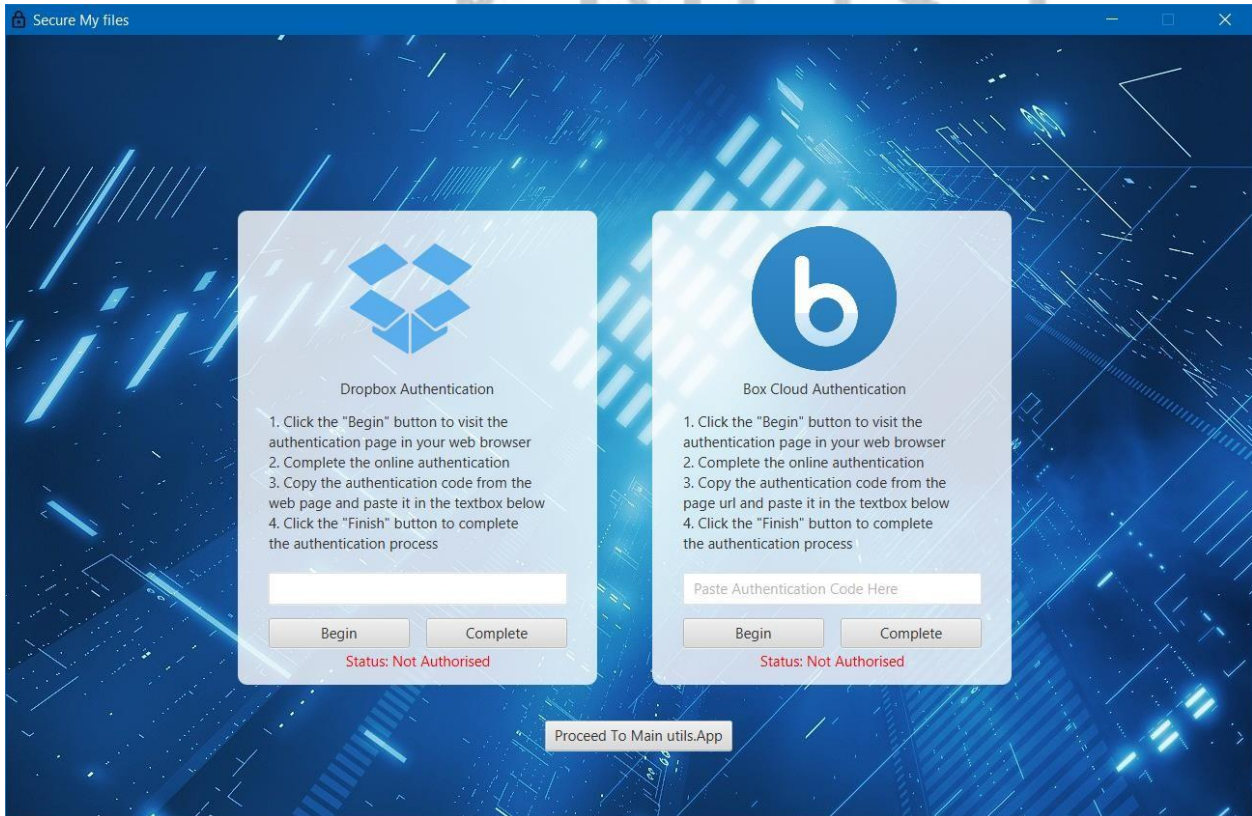


Figure 11: Cloud page

Secure My Files requires OAuth 2.0 authentication from the cloud storage service providers. To sign into a cloud account, the user first selects the radio button of the service provider, then click the “Begin” button to trigger the default web browser to open and load the authentication page for the selected cloud storage service provider. The user proceeds to grant authorization to the application.

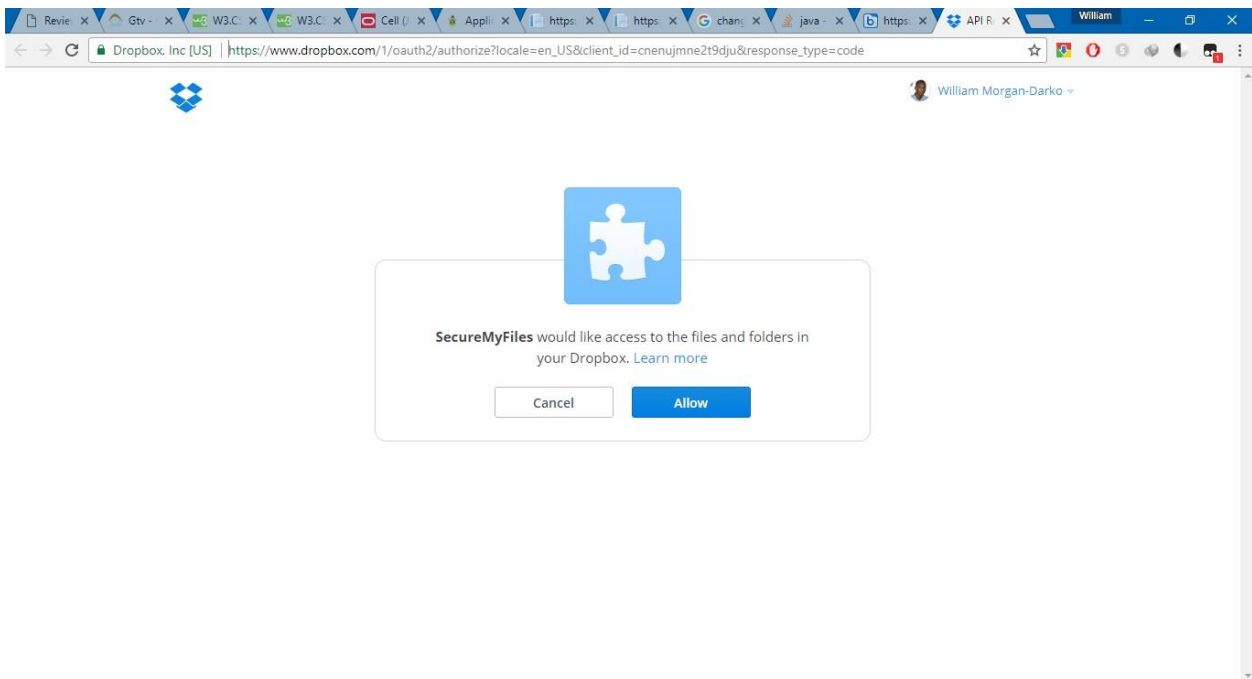


Figure 10: Authorization for Dropbox Cloud Storage service provider

The service provider presents an authorization code which the user must then copy and paste into the Secure My Files application.

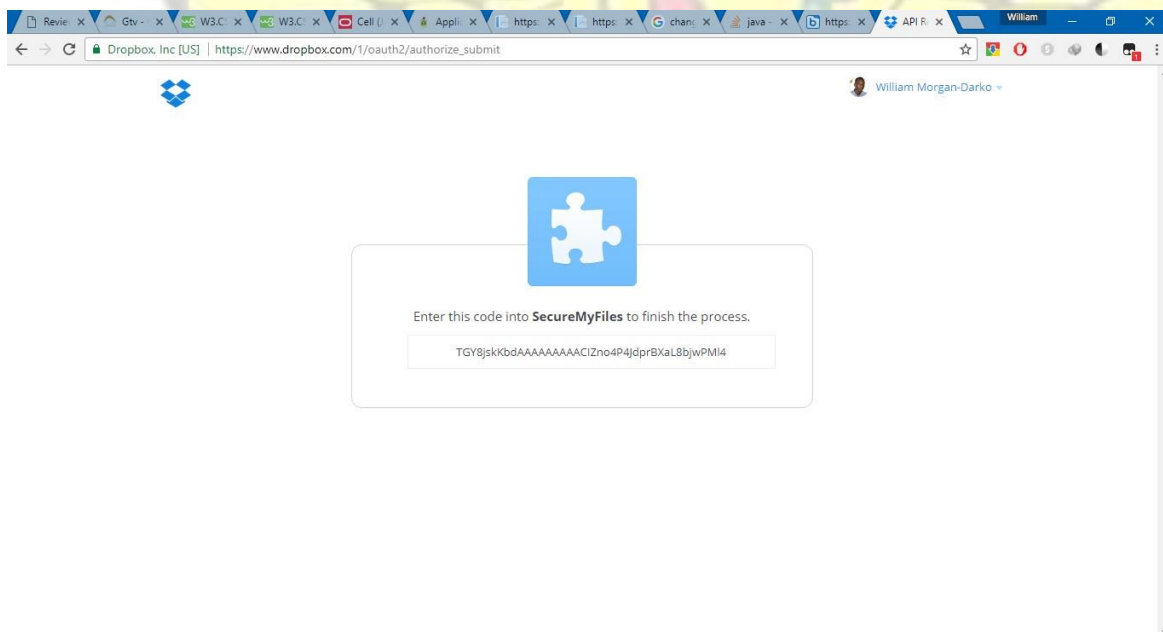


Figure 11: Sample Authorization code from Dropbox

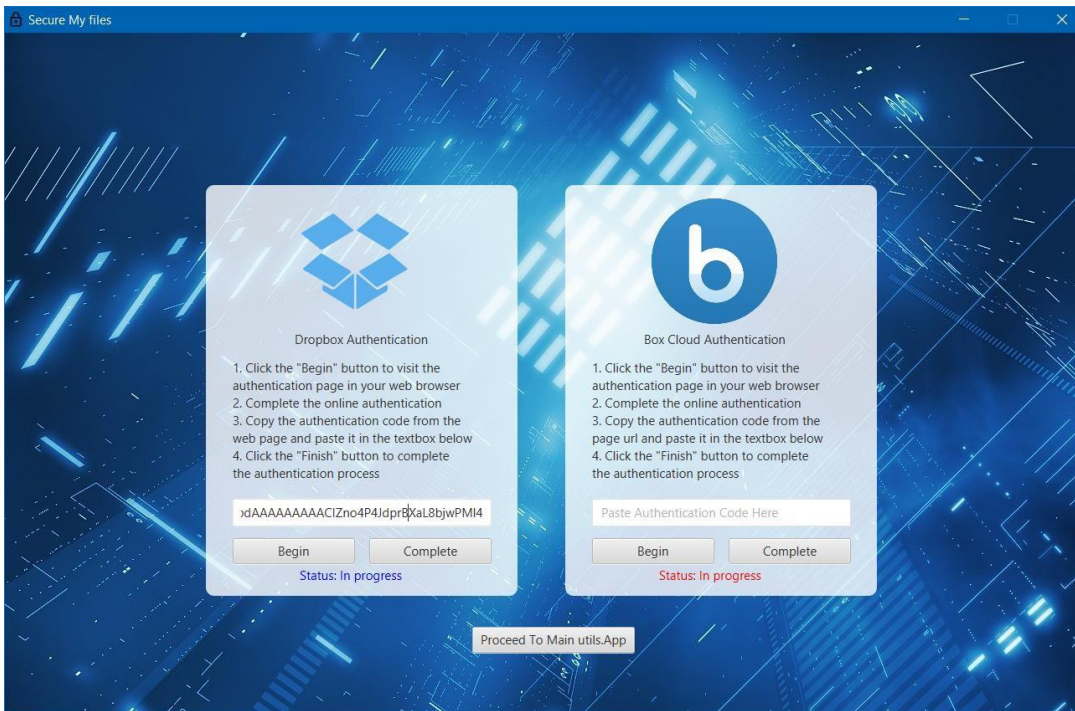


Figure 12: Cloud page with authorization code from Dropbox

The user completes the signing in process by clicking the “Complete” button.

Download a file

The user initiates a file download by first selecting one of the uploaded files from the list of file names on the main page. Then the user clicks the download button to trigger the processes which download the file shard, join them into a single file, decrypt them and rename the file from its hash value to the original file name.



Figure 13: Download button

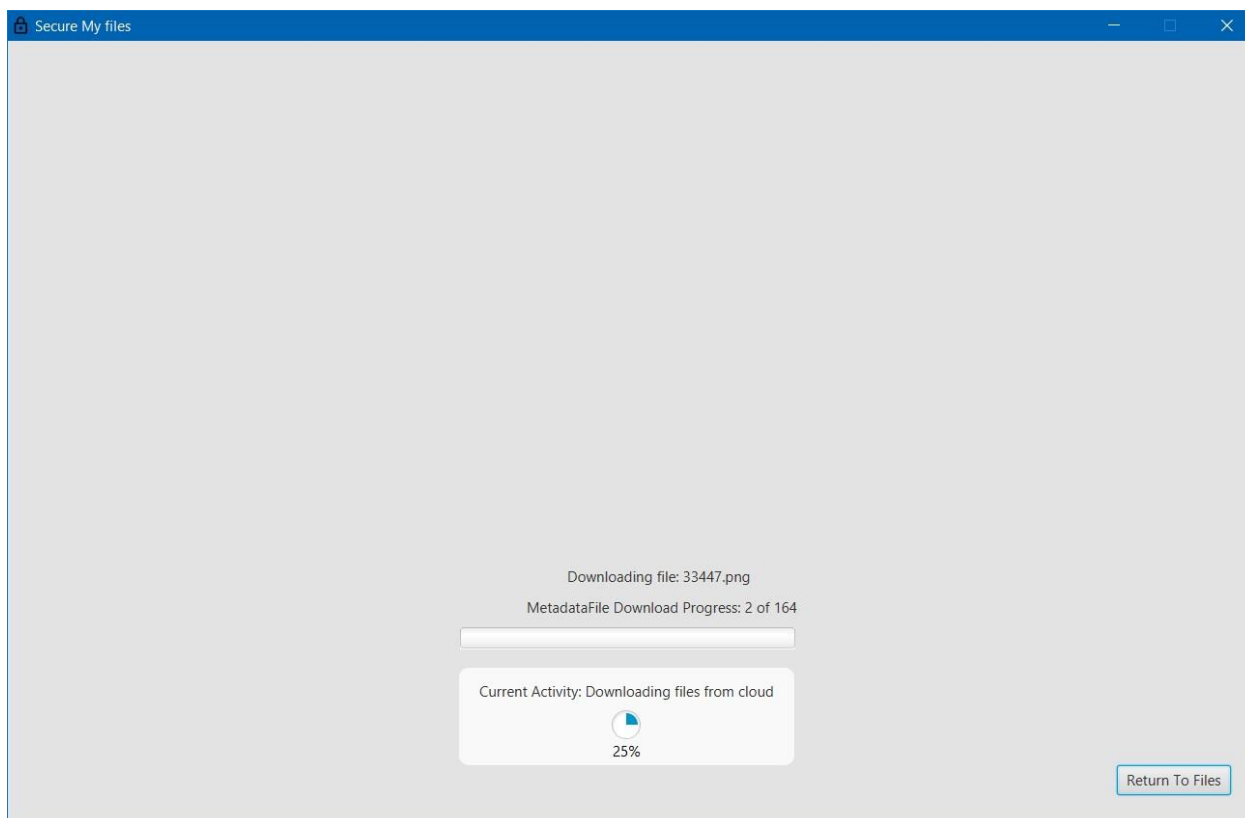


Figure 10: Download page showing the download of a file named “33447.png”

APPENDIX 3

Published articles from the study and reviewers comments.