Secure Cloud Storage

by

Jeff Yucong Luo

A thesis
presented to the University of Waterloo
in fulfilment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2014

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

The rapid growth of Cloud based services on the Internet invited many critical security attacks. Consumers and corporations who use the Cloud to store their data encounter a difficult trade-off of accepting and bearing the security, reliability, and privacy risks as well as costs in order to reap the benefits of Cloud storage. The primary goal of this thesis is to resolve this trade-off while minimizing total costs.

This thesis presents a system framework that solves this problem by using erasure codes to add redundancy and security to users' data, and by optimally choosing Cloud storage providers to minimize risks and total storage costs. Detailed comparative analysis of the security and algorithmic properties of 7 different erasure codes is presented, showing codes with better data security comes with a higher cost in computational time complexity. The codes which granted the highest configuration flexibility bested their peers, as the flexibility directly corresponded to the level of customizability for data security and storage costs. In-depth analysis of the risks, benefits, and costs of Cloud storage is presented, and analyzed to provide cost-based and security-based optimal selection criteria for choosing appropriate Cloud storage providers. A brief historical introduction to Cloud Computing and security principles is provided as well for those unfamiliar with the field.

The analysis results show that the framework can resolve the trade-off problem by mitigating and eliminating the risks while preserving and enhancing the benefits of using Cloud storage. However, it requires higher total storage space due to the redundancy added by the erasure codes. The storage provider selection criteria will

minimize the total storage costs even with the added redundancies, and minimize risks.

Acknowledgements

There are a number of people without whom this thesis might not have been written, and to whom I am greatly indebted.

I would like to express the deepest appreciation to my supervisor, Professor Gordon B. Agnew for his time, expertise, guidance, criticisms, and support throughout my graduate studies.

Sincere thanks go to Cisco Systems, Inc. and the Natural Sciences and Engineering Research Council of Canada for their financial support and technical expertise.

I would like to thank Professor Lin Tan for her guidance and supervision in developing me as a teaching assistant, as well as her constructive criticisms of this thesis. Thanks go to Professor Ian Munro for his insights and criticisms as well.

I would also like to thank Professor Raouf Boutaba, whom introduced me to the research problems in Cloud Computing. I want to thank all of the professors whom have taught and guided me throughout my studies.

I would also like to express the deepest thanks to my beloved parents as well as my best friend for their love, encouragement, patience and support.

Thanks go to many friends past and present, whom have kept me accompanied throughout this exciting journey.

DEDICATION

I would like to dedicate this to the memory of my grandparents, whom have raised me as a child, mentored me as an adult, and taught me the most valuable lessons in life. They were brilliant and well respected in their professional careers. As grandparents, they were the most kind and nurturing grandparents one could wish for.

Table of Contents

List of Figures

List of Tables

List of Equations

CHAPTER 1: INTRODUCTION

In recent years, the rise of Cloud Computing has given internet users a host of freedoms never enjoyed before. One such freedom is the ability to store files on the Cloud through a Cloud storage service provider, and retrieve it anywhere else in the world when the user authenticates to the service. It is increasingly being used as a repository for storing back up data. In team settings, Cloud storage lets teams synchronize and organize all kinds of shared data. For start-ups and small corporations, the use of Cloud storage in conjunction with Cloud Computing platforms reduces the need to invest in hardware equipment up front, allowing many new ideas to be developed into full scale products and large corporations. While Cloud storage services are numerous and on the rise, there are still many security, economic, and reliability issues associated with utilizing the Cloud as a storage medium.

Attacks such as the one aimed at Dropbox as recent as 2011 have allowed anyone on the internet to download any file stored and hosted by Dropbox for a 4 hour time period [1]. The attackers exploited a bug in Dropbox's authentication mechanism. Another attack aimed at Amazon S3 storage service in 2011 exploited vulnerabilities within the authentication mechanism of Amazon.com, which also allowed researchers access to data stored in S3 [2]. Cloud storage service providers remain a high value target for many attacks as the general public and users of the internet are unaware of the potential risks of using a Cloud storage service.

Cloud storage providers encounter numerous problems in their operation that results in service outages or even data loss for its users. These include power outages, natural disasters, hard disk or server failures, maintenance work, and administrative mistakes. Users who store data on the Cloud must depend on reliable storage services. As such, reliability remains a key concern for using a Cloud storage service.

Cloud storage service providers employ tier-based pricing to charge users for storing files on their Clouds. Most providers offer an initial free storage tier with a capacity limit that usually suffices for individual consumers, but companies will quickly outgrow the capacity limit. The lowest surveyed storage cost is at $0.05 per GB per month, offered by Google Drive. For consumers and companies, there are also costs and bandwidth limitations associated with internet service providers since almost all Cloud services are accessed through the internet. The lowest surveyed home internet connection cost is $0.10 per GB transferred. Competition between storage providers leads to a marketplace with ever changing storage pricing. Some storage providers also charge a fee to download data off its Cloud. To take advantage of lower long term storage costs, users must pay a relatively expensive fee in order to download and upload their data from an expensive storage provider to a cheaper provider. Literature refers this as a storage vendor lock-in, where it can be cost prohibitive to move away from a storage provider once a user or organization commits to using the services of that provider. Optimizing the total storage costs alone is a challenging dynamic problem, but coupled with the security requirements and reliability requirements it becomes a very challenging problem.

The central contribution of this thesis is a Framework for resolving the security and economic problems of using the Cloud as a storage medium while preserving all of its benefits. The Framework is a high level design of a secure storage system for the Cloud, from a consumer point of view. It is meant to be a guiding template for software designers whom wish to design and implement a secure storage platform or system. The Framework adopts the approach by Abu-Libdeh, Princehouse, and Weatherspoon [3] of using erasure code algorithms to split and join files to add a layer of reliability and security to the files. The thesis presents an in-depth analysis of the approach, comparative analysis of applicable erasure code algorithms, and the design and analysis of the Framework. The Framework provides a number of improvements upon the work by Abu-Libdeh et al.

1.1 Thesis Roadmap

The thesis is written in a manner that is hopefully accessible and interesting to a broad range of readers from different backgrounds, including internet users, network and security researchers, and software systems designers. Background knowledge of statistics at the level of a second year university course is helpful in understanding parts of the thesis, although not essential to digest the main ideas.

Internet users will want to begin their exploration at Chapters 2 and 3 to gain an in-depth understanding and appreciation of the technology and issues surrounding Cloud Computing and Cloud Storage, and the value of data. It also prepares readers new to the field with the knowledge of the security principles that cryptologists and network security specialists use every day to design secure systems. Network and security researchers will enjoy Chapter 5 and 6, which shows how erasure codes can be used to solve the problem at hand along with their performance and security

properties. Software system designers will enjoy Chapters 6, and 7 which present the Framework and compare it against traditional paradigms of remote data storage.

1.2 Chapter Descriptions

This thesis is organized into 8 chapters as follows:

- Chapter 2 introduces Cloud Computing, Cloud Storage, Secure and Reliable Storage Principles, and the Value of Data.
- Chapter 3 presents a thorough examination of the benefits, risks, and costs of using the Cloud as a storage medium.
- Chapter 4 presents a concrete problem statement for using the Cloud as a storage medium.
- Chapter 5 presents the approach and research work by Abu-Libdeh, along with the author's analysis of how the approach resolves the economic, security, and reliability problems, and how it retains the benefits of using Cloud as a storage medium.
- Chapter 6 presents the Cloud storage Framework, an in-depth analysis of various erasure codes which can be used in the Framework, a storage provider selection algorithm for the Framework, and a method for handling the metadata used within the Framework.
- Chapter 7 compares this Framework to traditional remote data storage paradigms.
- Chapter 8 draws the conclusions of the thesis and presents future work.

CHAPTER 2: BACKGROUND

Cloud Computing has an interesting history of sparse and sporadic development, which have only come together in its modern incarnation starting in year 2000. Like other important technologies that have defined and revolutionized computing, Cloud Computing is steeped in fundamental works dating as far back as the 1950s in the formative stages of computer science. In time, new generations of scientists and engineers built upon prior work to cause the right and necessary conditions for Cloud Computing to birth in the 21st century. Like other computational methodologies, Cloud Computing brings unique values to the table and has its equal shares of security challenges. In modern times where the internet is experiencing exponential grown in the amount of data it is receiving and processing every day, the security issues become ever more important. The value of data also grows, as more data are being mined, analyzed, and reduced into useful knowledge. This chapter introduces readers to Cloud Computing, Cloud Storage, Secure and Reliable Storage Principles, and the Value of Data to help readers establish a broad context and understanding of the technology, its value, and the security challenges in modern day Cloud Computing. We also introduce Finite Field mathematics in the last section of the chapter, to help prepare readers for the technical discussions later in the thesis.

2.1 CLOUD COMPUTING

Cloud Computing is the paradigm of internetworked computing whereby vast amounts of computing resources are pooled together and subdivided into units of resources to service user and workload requests, on demand. The types of resource units include virtual machines created inside powerful computation servers used

mainly for computational work, virtual storage created in storage servers for data storage and retrieval, and virtual networks created among the servers and network equipment to facilitate private and secure communications among participating virtual servers. The principle feature of Cloud Computing data centers which separates it from a classical data center is its ability to scale up and down to match the number of requests on demand. This is accomplished by dynamically allocating or de-allocating the resource units according to the arrival and completion of user requests.

Two key enabling technologies for Cloud Computing is the surge of availability of fast internet access by every day consumers, and the continuous hardware innovations which reduced the price for server hardware while increasing its computational capacity. Without fast internet access, sending and receiving the types of data to the Cloud, and within a Cloud would have been too slow to be useful to businesses and consumers. Without price reductions and compute capacity increases, Clouds could not service billions of users and requests every day.

The central idea behind Cloud Computing began in the 1950s during the era of mainframe computer systems [4]. These systems had all of the computation equipment arranged in a single server room, while employees accessed them via a central terminal. To efficiently utilize the system, time sheets were used to allow employees to reserve time on the system. Employees would coordinate their access to the system through the time sheets, thus sharing the resources of the system. The notion of sharing computation resources began in this era. Technically, the mainframes of this era were standalone systems which operate in isolation. True time-sharing capable mainframes would arise later in the 1970s.

Professor John McCarthy, inventor of the LISP programming language, gave a special lecture in 1961 at Massachusetts Institute of Technology where he remarked that "If computers of the kind I have advocated become the computers of the future, then computing may someday be organized as a public utility just as the telephone system is a public utility…. The computer utility could become the basis of a new and important industry" [5]. In essence, Professor McCarthy defined the notion of utility computing in that lecture, forever setting a goal and direction to bring computing to the public in an affordable way.

In the 1970s, IBM released an operating system called "Virtual Machine" for their System/370 mainframe systems [6]. This operating system allowed distinctive computation environments in every virtual machine, for every employee who connected to the system. The specific technique that IBM engineers and scientists developed was Dynamic Address Translation, used to translate a relative storage address per VM to a physical storage address on the storage mediums of the day. Virtualizing the storage was sufficient for the computer architectures and systems in those days to create independent computation environments in a mainframe. The idea behind Dynamic Address Translation of using a logical addresses in software, and translate it into physical addresses by hardware was of such value that today we can see its use in many modern computer systems. This marked the beginning of the era whereby computers could work on multiple tasks in parallel while maintaining independent computation environments for its users. The notion of "Virtual" in Virtual Machines was born.

Amazon Inc. is arguably the company who invented modern Cloud Computing, in the early 2000s. Their efforts aimed to improve the internal resource utilization of their

massive data centers powering Amazon's worldwide commerce web system. The results proved very successful, leading Amazon to rent out their extra computational capacities to the public in 2006. Whilst Amazon did not invent the notions of resource sharing, utility computing, or virtual machines, their work in defining the architecture of a modern Cloud lead to a concrete definition of the economic and pricing models for monetizing shared computation capacity.

Modern Cloud storage began as an evolution of classical web storage services (such as FTP servers), network-attached storage technology, and the virtual storage systems by IBM in the 1960s. FTP servers provided a method to transfer files to and from a remote FTP server, while network-attached storage provided a method to transfer files to and from a storage device located within a local or enterprise network. FTP, more specifically its secure variant SFTP, gave Cloud storage its first communications protocol for securely transmitting a file over the internet. Network-attached storage protocols enabled Cloud storage providers to network together thousands of hard drives and manage them in one central server to provide the storage service. Modern virtual machine technology development provided the notion and idea to dynamically allocate storage resources. Combined together, the idea of a dynamically allocated secure storage service on the internet was born.

The earliest modern Cloud storage was built by Amazon as part of the Amazon Web Service in 2006, which provided Cloud storage and computation services [7]. This service was available for everyone in the public to use. Since then, numerous competing services have been built. Some are built on top of Amazon, such as Dropbox, while others are built from the ground up, such as Microsoft OneDrive.

2.2 Secure and Reliable Storage Principles

The previous section hints at specific security principles for storing data. This section describes these principles in concrete detail, and outlines why they are important to satisfy when designing a secure Cloud storage system. The overarching principle is to "not put all the eggs in one basket".

2.2.1 Replication and Redundancy

If there exists only a single true copy of some data, and this copy was somehow lost, then the data would be lost with it forever. Replication involves "carbon" copying the original data to create backup copies. In digital file systems, this is relatively easy as the bits constituting the data are simply duplicated. Replication safe guards the data from being lost or destroyed, so long as at any time there are always at least two copies of the data in existence. If any one of these copies were to be lost, tampered with, or destroyed, we can simply create another copy from the intact copy of the data. Redundancy is a measure of how much duplication exists for some data. Generally, some redundancy is needed to ensure the safety of the data in the event of data loss or corruption. Redundancy viewed as a necessary cost, and the objective of this principle is to provide the required amount of data security at the lowest possible cost, namely, to minimize redundancy.

We define Resultant Size Factor as the resultant size of a file (after adding redundancy) divided by the original size of the file. If the original file was divided into K number of equal size pieces, and we add R number of redundancy pieces to the file whereby the size of each R piece is the same as the size of each K piece, then RSF is the total number of pieces of data (N) divided by the number of pieces of data constituting the original data (K).

Equation 1 – Resultant Size Factor

$$Resultant\ Size\ Factor = \frac{K+R}{K} = \frac{N}{K}$$

A file will always have at minimum a Resultant Size Factor value of 1.0, where R = 0, implying the file has no redundancy. As well, K is always at least 1. Logically, if the original data was not split, we would still have 1 piece of data. We also define Redundancy Factor as R divided by N:

Equation 2 – Redundancy Factor

$$Redundancy\ Factor = \frac{R}{N} = \frac{R}{K+R}$$

Redundancy Factor represents a relative level of security for some data. RF has a minimum value of 0.0, with R = 0, representing no redundancy. The higher the value of RF, the more secure the data is against data loss and corruption.

Redundancy Factor and Resultant Size Factor are interrelated. To minimize Resultant Size Factor, we have to minimize the Redundancy Factor. Both RSF and RF have no finite upper bound in its possible value, but the larger values, the more redundant the resultant data is. The goal of a Cloud storage system is to minimize Resultant Size Factor, getting it as close to the value of 1.0 as possible, with respect to a desired Redundancy Factor chosen as a design goal of the system. The thesis presents a number of techniques and algorithms towards this goal in Chapter 6. We set the redundancy factor to be equal among the algorithms to represent an equal level of security against data loss and corruption, and then compare their resultant size factor to determine how efficient the algorithms are at achieving this objective.

If we rewrite Equation 1 and Equation 2 with respect to N we have the following identities:

$$RSF = \frac{N}{K}$$

$$N = K \times RSF$$

$$RF = \frac{R}{N}$$

$$N = \frac{R}{RF}$$

Equating N on both sides, we can write RSF and RF as a function of each other:

$$K \times RSF = \frac{R}{RF}$$

$$RSF = \frac{R}{K \times RF}$$

$$RF = \frac{R}{K \times RSF}$$

Further, we can rearrange the identity such that both factors are on one side of the equation, forming the redundancy minimization function:

Equation 3 – Redundancy Minimization Function

$$RSF \times RF = \frac{R}{K}$$

If we hold RF at a constant value, then to minimize RSF we would require the algorithms to give us a lower ratio between R and K, implying essentially that a more efficient algorithm will be able to achieve the same RF while adding less redundant data pieces.

2.2.2 Confusion and Diffusion

In cryptography and cryptosystems research, the principles of confusion and diffusion are central in evaluating the strength of cryptographic algorithms and systems in safeguarding data. In this research area, plaintext refers to the data we want to safeguard through an encryption system that uses an encryption key (such as a password) to translate the plaintext into ciphertext. Ciphertext is the encrypted version of the plaintext. For example, if the data we wish to encrypt is an English essay, the ciphertext would appear to be a random collection of incoherent letters and symbols. In cryptosystems, attackers ultimately have the objective of obtaining the plaintext. The easiest of the three pieces of data to obtain by an attacker is the ciphertext as an attacker could listen in to a secured communication channel such as a free Wi-Fi hotspot. However, bad computing habits such as reusing the same password across multiple accounts quite often allows attackers to have easy access to the encryption key as well.

Originally defined by Claude Shannon in his paper “Communication Theory of Secrecy Systems” in 1949, Confusion refers to “making the relationship between the key and the ciphertext as complex and involved as possible” [8]. This is so that if an attacker obtains the ciphertext, they would need to spend significant effort to find out the relationship between the ciphertext and the key, and thus obtain the key. If the Confusion principle is not applied, then an attacker could easily obtain the key through the ciphertext, and then use the cryptosystem to obtain the plaintext.

Shannon also defines Diffusion in his paper as the effect of “dissipating the statistical nature of the plaintext over all ciphertext” such that the two statistics cannot be correlated. In modern digital cryptosystems, this means to spread the

effects of each bit of plaintext and key to as many bits of the ciphertext as possible. Ideally, every bit of plaintext, and every bit of key is involved in creating a single bit of ciphertext. That way, if any bit changes in the plaintext or key, the ciphertext would change completely. This aims to disperse any statistical characteristics of the plaintext and key over the entirety of ciphertext, so that attacks based on statistical methods become useless. If the Diffusion principle is not applied, an attacker can potentially determine the plaintext through the ciphertext without needing to know the key, by directly inferring the plaintext using the statistical characteristics of known languages such as English.

In practice, cryptographic algorithms utilize character substitution and character position permutation to achieve the principles of confusion and diffusion, respectively. Often multiple rounds of substitution and permutation is performed, such as in the DES algorithm, to ensure high cryptographic strength.

Cloud storage algorithms and systems would ideally apply both of these principles in tandem to safeguard users' data. The application of these principles could be accomplished by the use of encryption algorithms prior to the use of replication and redundancy algorithms, or as shown in Chapter 6, could be accomplished as part of the replication and redundancy algorithms. When an algorithm applies the Confusion principle, it has the **Confusion Property**. When an algorithm applies the Diffusion principle, it has the **Diffusion Property**.

2.2.3 Off-site Data Protection

Utilizing off-site data protection allows users to safeguard their data in the event that the onsite copy becomes corrupt, lost, or destroyed by unexpected events such as

natural disaster or complete system crash. The aim of off-site data protection is to allow for quick recovery from such disasters, by being able to retrieve the critical data from the off-site and use it to restore the onsite systems. The principle works upon the notion that major disaster events are unlikely to occur in both the onsite and off-site locations at the same time, relatively speaking, compared to the likeliness of only one of the two locations experiencing the same event.

A simple way to show this is a coin toss experiment, whereby we denote that if the coin tosses with Head side facing up, then a disaster occurs. Similarly, if it tosses with Tail side facing up, then a disaster does not occur. Coin 1 denotes the onsite, while coin 2 denotes the off-site. We can see that a coin tosses with probability of ½ for Head and ½ for Tail. Therefore, the individual probability of either one of the sites experiencing a disaster would be 50%. The two events are always independent of each other. The probability that both of the sites would experience disaster at the same time is equivalent to the probability that both coins would toss with Head side facing up at the same time, which is ½ × ½ = ¼. Generally, the joint probability is shown as follows:

Equation 4 – Joint Probability of Two Statistically Independent Events

$$P(A \text{ and } B) = P(A) \times P(B)$$

Events such as natural disasters and system crashes have their own probabilities, and are often less than 50%. Equation 4 shows that no matter what the individual probabilities are, the chance of two independent events occurring at the same time will always be less than the chance of them occurring individually.

Using off-site data protection, a user can spread and reduce risks due to disasters and unexpected events by taking advantage of this principle. Further, a user can utilize multiple off-sites to protect their data, further reducing the risks. Cloud storage systems would ideally take advantage of this principle by utilizing multiple off-sites to safeguard users' data.

2.2.4 Principle of Least Privilege

The principle of least privilege states to grant a user, task, process, or server only the information it needs to accomplish its legitimate tasks. A Cloud storage environment contains the data of many users at a time, often in shared hardware and computing environments. The principle needs to be enforced to safeguard users from being able to modify and access each other's data when permissions have not been granted. Users should by default only be granted access to their data and no one else's. If a user shares some data with another user, then that user should only be able to have access to the shared data and not the private data of other users. Users should be prevented from being able to obtain administrative access and privileges, and thus other users' data. Likewise, the design of the Cloud storage system should be in such a way that the data is always protected from access until a user proves beyond a doubt they are who they claim they are. That is, the privilege of access is no data, by default.

2.3 Value of Data

A person's data is intrinsically valuable. Human activities on the planet have generated enormous amounts of data which, relative to history, have only been recently studied en masses due to the recent surge of Big Data analytics and software. A single individual's data can range from personally identifiable

information such as health records, private data files, publicly available birth records, to non-obvious data such as highway traffic information – an amalgamation of data of many individual's driving habits. All of these data are valuable. Some of them are valuable to the individual, while others are valuable to their families and friends. Some of the data is highly valuable for certain businesses but not to other businesses, and some of them are highly valuable to the government. This section explores in detail how much value data has.

2.3.1 Value of Personally Identifiable Data

Personal Information is considered by the vast majority of laws across the world as any information which can be used to uniquely identify an individual, whether living or deceased.

Identity theft is one of the most worrisome problems related to the loss of personal information. The Royal Canadian Mounted Police (RCMP) defines identity theft as the "collection and possession of someone else's personal information for criminal purposes" [9]. RCMP further defines identity fraud as the act of using someone else's identity to commit acts of fraud. Identity thieves aim mostly to steal and obtain access to someone's financial accounts and resources, such as bank accounts, credit card information, and passwords. Access to this information can deplete one's financial resources instantly, as well as cause damages to one's financial record such that the victim can no longer borrow funds from banks or obtain credit. Other main uses of such personal information include impersonation for the purpose of illegal entry, stay, and work, and tracking the person's movements.

Research by L. Sweeney at Carnegie Mellon University [10] shows that for United States of American citizens, 87% of them are uniquely identifiable simply through three pieces of data of mailing ZIP code, gender, and date of birth. The three pieces of data can be obtained as easily as looking through publicly available medical data, and voter list. Sometimes, personally identifiable information can be obtained through just a single piece of document such as a passport.

2.3.2 Legal Requirements of Personally Identifiable Data

In Canada, the Personal Information Protection and Electronics Documents Act (S.C. 2000) states the rules and conditions whereby personal information can be collected, stored, used, and disposed [11]. The act is often referred by its acronym PIPEDA. The act requires organizations to obtain informed consent by individuals prior to the collection of their personal information, and only for reasonable purposes that are clearly stated. It requires that the organization collect only the information in needs to fulfill the stated purposes, and no other personal information. It also requires organizations to safely store and protect the information with appropriate security measures against unauthorized access, disclosure, copying, use, or modification. Businesses must also destroy the information safely when it is no longer needed or if the business purpose for use of the information changes. The act also grants the individuals the right to see all the information collected about them and to correct the information if they are wrong [12] [13]. Exceptions to these rules are also stated in the act, such as that an organization is not obligated to disclose information of one individual if such disclosure would inadvertently disclose information about other individuals. Organizations however are obligated to disclose information when

the non-disclosure would obstruct justice and the enforcement of law, or compromise the safety of the persons in emergency situations [12].

An interesting violation of PIPEDA occurred in 2007, where Google's collection of street images in Canada for its Street View application captured many images of individuals with sufficient clarity to allow the individuals to be identified. Since PIPDEA considers such images to be personal information, the works were subjected to Canadian laws. A letter from Canada's Privacy Commissioner to Google states that Google collected the imagery "without the consent and knowledge of the individuals who appear in the images" [14], and that even though the Street View application allowed individuals to request images to be removed, "by the time individuals become aware that images relating to them are contained in Street View, their privacy rights may already have been affected". Google's solution came in the form of a slightly different version of Street View which adheres to Canada's privacy laws [15].

Australia requires personal information to be protected from "misuse, loss, and unauthorized access, modification, and disclosure" and has their own legislative requirements which Australian companies must follow [16]. In the United States of America, the Health Insurance Portability and Accountability Act (HIPAA) is a set of privacy laws regarding health records and health information of patients. Improper handling of a person's health record can result in fines as steep as $1.5 million USD per year [17].

While there are many laws and regulations in place around the world, a Cloud storage service provider will need to collect some amount of personal information in order to provide their services to those individuals; this includes financial information if the

service charges fees. It is important that the design of a Cloud storage system respects these laws and provide proper security in safeguarding personal information.

2.3.3 Business Value of Personal Data

Personal data can be immensely valuable to companies. Data trading companies alone thrive on a pure business model of buying and reselling a person's consumer behavior data. Much of the earnings in this industry come from the billions of dollars companies are spending advertising their products. Prior to the social network revolution, users often gave away uniquely identifiable personal information when they registered for an account for a website or web service. Such websites include online shopping websites, email services, instant messaging services, and others. The general types of information in this era was more explicit; for example, "what's your marital status?" might be a question asked during a user account sign up page. Otherwise, the type of data can be explicitly derived from a person's trail of activities on a website; such as suggesting products through purchase history on Amazon.com.

The sources of data and patterns of activities increased as a result of social networking. For example, if two people change their relationship status with each other on an online social networking site to "married", such information is of great value to any company involved in making and selling baby products. Such data is often used by advertisement companies to present products and services to users at such times. Such information could be obtained today even without an explicit change to an account profile, for example by utilizing image classification algorithms to data mine photo albums for wedding dresses and suites and faces to infer that a marriage happened between two people. Facial recognition is already a highly

successful feature of many social networking sites, and it would be inevitable to see more of such algorithms, such as DeepFace used by Facebook [18].

From repositories like these, a company can mine the shopping and communications habits and data of all their users and derive immense knowledge of them. There is an entire data trading industry for personal data known as the data broker industry, with a worth in multibillion dollars [19]. This industry is a fast growing, but ultimately a subpart of an even greater marketing industry.

Certain companies make profit off of the data they collect directly by utilizing that data to deliver targeted advertisements to its users. Examples include Facebook, Google, Netflix, Amazon, Microsoft, and others. Economists tend to analyze these companies in bulk by taking the total revenue divided by total number of users as a simplified means of calculating the worth of each user's data. For example, in Facebook's 2013 year-end earnings report [20] [21], they've cited 757 million daily active users and revenue of $2.585 billion USD for fourth quarter 2013. Dividing the two numbers shows an average of $3.41 revenue per daily active user for that quarter. For the entire year, the revenue is $7.872 billion with an average daily user of around 712.25 million, making the revenue per user per year at $11.05 USD.

From a broader perspective, consumer behavioral and personal data fall under the global internet commerce economic model, where a plausible measure for the entire economy can be the total dollars spent on marketing products and services to consumers. After all, companies do want to influence and capture an individual as a customer. There are also other markets, such as health care, whereby a person's data would likely be traded for money since such data would be highly useful in health

care research and health care products market research. It is sufficient to conclude that a person's data can be worth a lot.

2.3.4 Costs From Loss of Data

Whenever a company loses personal information and data, a multitude of consequences usually follows. Generally it can result in legal fines, financial loss, loss of intellectual property, loss of customers, and most critically loss of trust. Without a trustworthy reputation, a company will have a hard time conducting business.

One of the most exemplary cases of loss of customer information was by Sony Corporation on April 16th and 17th, 2011 [22]. Sony Online Entertainment (SOE)'s press release states that up to 24.6 million customer account information might have been stolen by criminals, including names, addresses, email, birth date, gender, phone number, login names and hashed passwords. Within these, 12,700 credit or debit card numbers and expiration dates were stolen, as well as 10,700 debit records. A letter from Sony's Chairman to the US House of Representatives, states that the PlayStation Network (PSN)'s 77 million registered accounts were also affected [23].

Sony took down all of their online gaming services to fix the security issues. They granted every customer 30 days of free subscription time for their online gaming network services, as well as a free day for each day their system was offline. Given that the price for a month of Sony Online Entertainment subscription time was $14.99 USD back in 2011 [24], and the 23 day closure of the PSN and SOE networks [25], this is equivalent to giving away $651.47 million USD of free online gaming services. Author was unable to find the cost of PSN monthly subscription costs, but can online imagine the free service cost figure to rise even higher. News reports also

indicated that Sony expended $171 million USD to conduct forensics investigations, repair the services, and perform other duties related to this breach [25].

Another exemplary case occurred in June of 2012 where, allegedly, an employee working for the Alaska Department of Health and Social Services Department (DHSS) lost a portable electronic storage device containing electronic health records. This case was investigated by the U.S. Department of Health and Human Services, resulting in a fine of $1.7 million USD for Alaska's DHSS [26].

2.4 Finite Fields

A light background in Finite Field mathematics is required for understanding Section 6.3.7 of this thesis. This section presents an introduction to Finite Field mathematics. Finite Fields are also called Galois Fields, named after its inventor Évariste Galois whom published it in 1846 as "Œuvres Mathématiques" (English: "Mathematical Works") in the *Journal de Liouville* [27], and subsequently republished by *Societé Mathématique de France* (English: Mathematical Society of France) in 1897 [28].

As defined by Menezes et al in their *Handbook of Applied Cryptography* [29], a Ring (**R**, +, ×) consists of a set **R** with two binary operations + (addition) and × (multiplication) on **R**, where it satisfies the four conditions:

1) (**R**, +) is an abelian group with identity denoted **0**, that is it is closed, associative and commutative for the + operation.
2) The operation × is associative, that is $\mathbf{a} \times (\mathbf{b} \times \mathbf{c}) = (\mathbf{a} \times \mathbf{b}) \times \mathbf{c}$ for all $\mathbf{a}, \mathbf{b}, \mathbf{c} \in \mathbf{R}$.
3) There is a multiplicative identity denoted **1**, with $\mathbf{1} \neq 0$, such that $\mathbf{1} \times \mathbf{a} = \mathbf{a} \times \mathbf{1} = a$ for all $\mathbf{a} \in \mathbf{R}$.

4) The operation × is distributive over +. That is **a** × **(b** + **c)** = **(a** × **b)** + **(a** × **c)** and **(b** + **c)** × **a** = **(b** × **a)** + **(c** × **a)** for all **a, b, c** ∈ **R**.

A Ring is a commutative ring if **a** × **b** = **b** × **a** for all **a, b** ∈ **R**. Each element **a** of a ring **R** is called a *unit*, or an *invertible element* if there is an element **b** ∈ **R** such that **a** × **b** = **1**. In this case, **b** is the multiplicative inverse of **a**.

A Field is a commutative Ring in which all non-zero elements have multiplicative inverses. A Finite Field is a field **F** which contains a finite number of elements. The order of **F** is the number of elements **C** in **F**. The number of elements must be a prime power, that is $\mathbf{C} = \mathbf{P}^{\mathbf{M}}$, where **P** is a prime number. If **M** = 1, the fields are called Prime Fields. If **M** ≥ 2, the fields are called Extension Fields. A Finite Field is denoted by the notation GF(C), shorthand for GaloisField(C). For example, Figure 1 shows a Finite Field of $2^4 = 16$ elements.

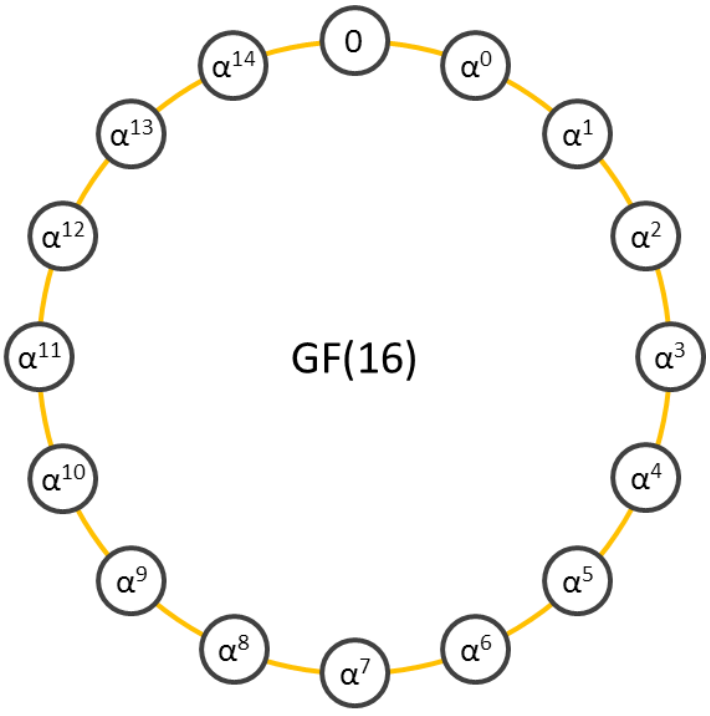

Figure 1 – Galois Field of 16 Elements

The elements of the Finite Field are based upon a primitive element α, taking on the values 0, α^0, α^1, α^2, α^3, α^4, … α^{C-1}. α^0 always equals 1. This set also forms a notation known as the Field's Index Representation. For a given Finite Field, α^N is the index of the element, while the element would contain some particular value depending on what α is.

2.4.1 Finite Field Generator Polynomial and Representation

A Finite Field has a particular property called the Field Generator Polynomial, denoted as G(X). It is a polynomial of degree M which is irreducible. A polynomial is irreducible if there are no factors with coefficients over the integers Z. The only polynomial and element that is irreducible in GF(2) is 1, thus the field generator polynomial for GF(2) is G(X) = 1. For GF(16), two polynomials are irreducible: $X^4 + X + 1$ and $X^4 + X^3 + 1$. The rest of the examples in the thesis use $G(X) = X^4 + X + 1$ as the generator polynomial for GF(16).

Besides the Index Representation shown earlier, each element of the Finite Field is also represented by a polynomial in the form $A_{M-1}X^{M-1} + A_{M-2}X^{M-2} + A_{M-3}X^{M-3} + \ldots + A_1X + A_0$. Mathematically this polynomial is related to the primitive element α by the field generator polynomial. The Generator Polynomial generates the values of each element. Further, a defined property is that $G(\alpha) = 0$ as the primitive element is a root of the generator polynomial.

2.4.2 Binary Fields and Polynomial Representations

In digital systems, Binary Fields are used. Binary Fields are $GF(2^M)$ fields, where M represents the number of bits of any element in the field. The most elementary

Binary Field, GF(2), has only 1 bit and two elements of 0 and 1. In binary fields, the primitive element α is 2.

In GF(2), addition is the logical XOR of the two bits and multiplication is the logical AND of the two bits. There's no bit carry or borrows in this field, as such subtraction and division are the same as addition and multiplication, respectively. A Computation table is provided for GF(2) below in Table 1.

Table 1 – Finite Field Arithmetic for GF(2) Binary Field

+	0	1
0	0	1
1	1	0

×	0	1
0	0	0
1	0	1

−	0	1
0	0	1
1	1	0

÷	0	1
0	0	0
1	0	1

In Binary Fields, an element can be represented by its binary value: $A_{M-1}A_{M-2}A_{M-3}\ldots A_1A_0$. For example, the element 5 can be represented as {0101} in binary form in GF(16). Elements could also be represented in a polynomial expression in the form: $A_{M-1}X^{M-1} + A_{M-2}X^{M-2} + A_{M-3}X^{M-3} + \ldots + A_1X + A_0$, where the coefficients A_i are the binary values of the element. In GF(16), the polynomial representation is: $A_3X^3 + A_2X^2 + A_1X + A_0$. The element 5 would be represented as $X^2 + 1$.

Using G(X) from 2.4.1, we can substitute in the primitive element α to obtain the following equivalencies:

$$G(\alpha) = \alpha^4 + \alpha + 1 = 0$$

$$and$$

$$\alpha^4 = \alpha + 1$$

Note: Additions and subtractions are the same in a binary field; this is shown in Section 2.4.3.

To enumerate the polynomial representations for each of the elements of a Finite Field, one uses the generator polynomial to compute α^N for N = 0 to C-1, reducing the polynomial via substitution. Eg: $\alpha^4 = \alpha + 1$ from above, and $\alpha^7 = \alpha^6 \cdot \alpha = \alpha^4 + \alpha^3 = \alpha + 1 + \alpha^3$. Using G(X) above, the representations for GF(16) is shown below in Table 2.

Table 2 – Field Elements for GF(16) with G(X) = $X^4 + X + 1$

GF Index	Reduced Polynomial Form	Decimal Value	Binary Value
0	0	0	0000
α^0	1	1	0001
α^1	α	2	0010
α^2	α^2	4	0100
α^3	α^3	8	1000
α^4	$\alpha + 1$	3	0011
α^5	$\alpha^2 + \alpha$	6	0110
α^6	$\alpha^3 + \alpha^2$	12	1100
α^7	$\alpha^3 + \alpha + 1$	11	1011
α^8	$\alpha^2 + 1$	5	0101
α^9	$\alpha^3 + \alpha$	10	1010
α^{10}	$\alpha^2 + \alpha + 1$	7	0111
α^{11}	$\alpha^3 + \alpha^2 + \alpha$	14	1110
α^{12}	$\alpha^3 + \alpha^2 + \alpha + 1$	15	1111
α^{13}	$\alpha^3 + \alpha^2 + 1$	13	1101
α^{14}	$\alpha^3 + 1$	9	1001

Since α is 2 in Binary Fields, we can substitute that into the polynomial to get the equivalent decimal value of each element. We can also extract the coefficients from the polynomials to obtain their binary values, which match their decimal values.

2.4.3 Binary Field Arithmetic

When adding two elements in a binary field, their polynomials are added together:

$$C_{M-1}X^{M-1} + C_{M-2}X^{M-2} + \ldots + C_1X + C_0 = A_{M-1}X^{M-1} + A_{M-2}X^{M-2} + \ldots + A_1X + A_0 + B_{M-1}X^{M-1} + B_{M-2}X^{M-2} + \ldots + B_1X + B_0$$

where $C_i = A_i + B_i$, for $0 \le i \le M-1$.

Since the coefficients are binary numbers, they can only take on the values of 0 or 1, so they are added through the XOR operation. To simplify: $C_i = A_i$ XOR B_i, for $0 \leq i \leq M-1$.

For example, adding decimal values 10 and 9 (index values α^9 and α^{14}) in GF(16) using binary operations would be calculated as follows:

$$10 + 9 = \{1010\}\, XOR\, \{1001\} = \{0011\} = 3$$

An alternative way to add the two values is to directly add their index positions modulus G(X). G(X) in decimal value would be 16 + 2 + 1 = 19. Since $\alpha^9 = 10$ and $\alpha^{14} = 9$, their index positions are 9 and 14 respectively.

$$9 + 14 \bmod 19 = 23 \bmod 19 = 4, \text{ and } \alpha^4 = 3$$

Subtracting 9 from 10 would be calculated as: 10 – 9 = {1010} XOR {1001} = {0011} = 3. Thus subtraction and addition are the same in binary fields.

Multiplication in a Binary Field involves multiplying their polynomials modulus G(X). For example multiplying 10 to 9 in GF(16) would be calculated as follows:

$$10 \times 9\ Mod\ G(X)$$

$$= (\alpha^3 + \alpha) \times (\alpha^3 + 1) Mod (\alpha^4 + \alpha + 1)$$

$$= (\alpha^6 + \alpha^4 + \alpha^3 + \alpha) Mod (\alpha^4 + \alpha + 1)$$

$$= \alpha^2 + 1$$

$$= 5$$

Similarly, Division in a Binary Field involves computing the quotient of the two polynomials. Often this is done by long division procedure.

$$10 \div 9 = (\alpha^3 + \alpha) \div (\alpha^3 + 1) = 1$$

Table 3 summarizes the arithmetic operations in a Binary Field.

Table 3 – Binary Field Arithmetic

Operation	Calculation in Classical Algebra	Calculation in GF(2^M)
Addition	R = A + B	R = A XOR B in Polynomial Form
Subtraction	R = A – B	R = A XOR B in Polynomial Form
Multiplication	R = A × B	R = (A × B) Mod G(X) in Polynomial Form
Division	R = A ÷ B	R = Quotient of A ÷ B

2.5 Chapter Summary

This chapter introduced Cloud Computing and Cloud Storage as a highly scalable, utility computing paradigm with a rich history since the 1950s. Cloud Computing groups together vast pools of computation resources and dynamically allocates them to live user requests or workloads.

We also presented the key secure and reliable storage principles which cryptographers and system designers use to design modern secure systems. Replication and Redundancy helps safeguard data by making them resilient to losses. Confusion and Diffusion principles define the necessary properties a system must have to be considered cryptographically secure. Off-site data protection helps users protect their data against local equipment failures. Finally the Principle of Lease Privilege guides us towards designing secure access rules to ensure that users of a system cannot effect actions which they are not authorized to do so.

We explored the value of personal data from four perspectives. We showed the potential damage misplaced data could cause to users. We showed the legal requirements on personal information around the world, and the consequences when the law isn't followed by companies. We showed the revenue companies can make when they have legal access to personal information, and the costs companies pay when they failed to safeguard personal information.

Finally we introduced Finite Field mathematics.

The next chapter examines the risks, benefits, and costs of using Cloud storage.

Chapter 3: Cloud Storage Risks, Benefits, and Costs

The use of Cloud storage comes with unique risks, benefits, and costs through a new economic model. This chapter presents all three of these in depth to give readers a detailed understanding of the trade-off between the risks and benefits, and the costs to use Cloud storage. For the purposes of discussion, we consider data to be encapsulated in digital electronic documents, or simply files. Files can contain structured and organized information such as spreadsheets, or unstructured information such as books and videos.

We begin this chapter by presenting an in depth analysis of the risks of using Cloud storage in Section 3.1. Then we show the analysis of the benefits users can reap in Section 3.2. We show the costs associated with using Cloud storage along with a comparison to using local storage in Section 3.3, and finish the chapter with a summary.

3.1 Cloud Storage Risks

A user's data is highly valuable to both the user and to the organizations and businesses providing the storage service. This section examines the types of risks a user faces by storing files on the Cloud.

3.1.1 Malicious Attacks from Anywhere in the World

The ability to authenticate and access the data from anywhere around the world presents a unique problem and risk. Attackers no longer need to physically track

down the specific device or hard drive containing the data desired; instead they may concentrate their efforts at breaking the authentication mechanism to obtain vast troves of data from many users. The vast majority of Cloud storage system security breaches are related to authentication mechanism weaknesses or attacks, for example the 2011 attack on Dropbox [1] which allowed anyone on the internet to download any files stored and hosted by Dropbox for a 4 hour time period. Another example comes from a paid research by Amazon. In 2011, Amazon invited a team of security experts and researchers to conduct attacks on their servers. The researchers were able to access data stored in Amazon's S3 service [2]. In the 2009 attack against Twitter, the anonymous attacker exploited weaknesses in password recovery mechanism of Google's email service, and was eventually able to obtain many confidential corporate documents and information from email attachments of the corporate email accounts of Twitter employees, which was a hosted email service on a Cloud run by Google [30]. Breaches in the authentication mechanism of Cloud storage providers prove to be deadly in terms of allowing a user's private data be accessible and exposed to the entire world, and allowing it to be modified or deleted by attackers. An ideal solution would allow data to be safe even when authentication mechanisms have been compromised.

3.1.2 Implicit Dependence on Storage Provider Reliability

Storage providers can sometimes halt their services in order to perform periodic maintenance work to their systems, which presents a risk to users if users need access to their data during times of unavailability. Although it is not a security risk, this is concerning when users send files onto the Cloud and remove all local copies to maximize their storage space. There is also a general trend of moving computation,

software, and data storage completely to the Cloud, where local machines serve only as consoles to remotely access the software and data on the Cloud [31]. Access outages may also be caused by internet service providers, or by natural disasters. All of these potential sources of outages are not in a user's control, and in fact always has a probabilistic chance of occurring. No matter the source or reason, outages will cause inconvenience for users. An ideal solution would be able to work around outages.

3.1.3 Risk of Data Loss and Data Corruption

Major storage providers have software mechanisms in place to mitigate equipment failure [32], however there is always the chance that a user's data is completely lost. For example, natural disasters may flood or short circuit an entire data center, corrupting all of the data. Mistakes made by employees may misplace sets of hard drives during upgrade or maintenance, losing the data. Software mistakes may cause user's data to be written over. In fact, Clouds are utilizing cheap commodity hard disks as a means to minimize costs, which have higher risks and chances of failure compared to server grade hard drive equipment. Whenever complete data loss occurs on a Cloud, a user only has the option to re-upload the data to another more reliable Cloud, assuming the user has a local copy. An ideal solution would distribute a file among several Cloud service providers, so that the user can enjoy the benefits of Cloud storage and be able to tolerate complete data loss by individual Cloud service providers.

3.1.4 Implicit Requirement to Always Trust the Provider

The use of Cloud storage services requires users to implicitly trust the service provider. Users must trust the service provider's ability and capability to defend

against attackers and intruders, to safeguard the data against equipment failure, to not compromise and modify their data, and to respect the privacy and confidentiality of the user as well. While there are terms of use agreements, and privacy policies in place, the vast majority of Cloud storage providers ultimately disclaim any liability and responsibility for the data stored on their Clouds. It is a compromising position when the absolute control of a user's Cloud data rests in the hands of the service provider, but the absolute responsibility for the data rests on the user. The requirement of such implicit trust is often disregarded by users in lieu of the gains of the benefits of Cloud storage. In an ideal case, this requirement should be removed while still retaining the benefits of Cloud storage.

3.1.5 Conflicting Laws May Not Respect Users' Privacy

It is known that data stored on the Cloud has become a hot target for law enforcement and security agencies as they issue access for information requests and warrants to obtain data, often in bulk. Such warrants apply equally to local computers and storage devices, of course. Law abiding citizens would comply with the warrants when requested. The complexity starts rising when consideration is given to the fact that, quite often, Cloud services hosted by one country can and is used by users from all over the world. A bulk data request may inadvertently allow one nation's law enforcement obtain the data stored by a citizen of another nation, simply due to the physical location of the data center and Cloud. Many nations have laws in place regarding the placement and location of data in terms of their physical storage devices. For example, in British Columbia, Bill 73 – the Freedom of Information and Protection of Privacy Amendment Act, 2004 states "A public body

must ensure that personal information in its custody or under its control is stored only in Canada and accessed only in Canada" [33].

Most nations also have laws regarding any data physically stored within their geography. By default, a user has to assume that the data they place in a specific Cloud will be subject to all the local laws, regulations, culture, customs, and security standards of the nation in which the data is physically stored. The complex layers of national and international laws require solutions through political dialogue and international treaties, which is beyond the scope of this thesis. However, of importance to a user is an ability to know where their data is located geographically around the world. Better yet, users should be able to control where their data is located, so that they can avoid placement of their data in nations that they feel might be risky or not trustworthy.

3.2 Cloud Storage Benefits

There are numerous benefits to storing files on the Cloud, including economic cost reductions, flexibility, world wide access, and improving resource utilization. This section analyzes all the benefits of using Cloud storage in detail.

3.2.1 A New Economic and Business Management Model

Cloud storage and Cloud Computing both offer an economic model for consumers to pay only for what they use, like a utility bill. The infrastructure and operation costs are paid by the service provider, for example the costs for purchasing thousands of hard drives, equipment set up and maintenance, management, and electricity costs. In turn, these costs are shared among all users of the system, being usually charged a fixed rate for each unit of resource usage or consumption. Storage providers typically

offer various storage quotas, which can be increased or decreased on demand by users.

Since all installation and management work is performed by the storage provider, start-up companies and small to medium size businesses can take advantage of this to reduce upfront costs. For short term projects, they also don't have to worry about reselling any hardware. The economic flexibility and savings can allow companies to hire more staff to accelerate their ideas and projects to meet goals and milestones faster.

3.2.2 Improved Resource Utilization

From a resource utilization perspective, pooling together resources and users is a highly efficient means to consume the resources. Studies by the University of Pennsylvania have shown that computer machines consume on average 50% to 90% of electricity when they are idle compared to when they are fully loaded with computation tasks [34]. The idle power consumption rate depends largely on the computer manufacturer and whether the LCD was kept on while the computer idles. Naturally it makes sense to improve the utilization of servers by constantly assigning tasks to them to keep them active. Similarly, any unused portion of a hard drive may generally be seen as a waste of the resource. If a project requires 500GB of storage, then purchasing 1TB of storage space is unnecessary and wasteful, increasing the effective price per GB of storage. Prior to Cloud storage, users often purchase some additional space in their computers for use by temporary files, and to anticipate for any of their growing data storage needs. Cloud storage allows users to request for additional storage on demand, increasing the size of their allowance when they need

it. Similarly it allows users to reduce their allowance when they no longer need the extra space.

3.2.3 Worldwide Access

Users today carry portable computing devices with them on a daily basis, and access their data from many different machines. Cloud storage allows a user to authenticate to the service and access their data from any device which can operate the service's software. Users can also access their data from anywhere in the world as long as they have a connection to the service. The ease of access from across the world is a strength of the Cloud storage service, compared to more traditional means such as carrying around portable storage devices like a USB drive.

3.2.4 File Versioning and Recovery

Cloud storage services can also retain versions of a user's files and data through time, allowing the user to revert unintended changes, or mistakes such as accidentally deleting an important file. On traditional hard drives in a local computer, a user would have to remember what changes occurred and try to manually revert them if it is possible. For any deleted files, users would have to utilize disk recovery software to attempt to recover the data from the file. The former is error prone and relies upon human memory, while the latter has no guarantee of success because once a file is deleted the system treats the space taken up by that file as free space and might write over the data with data from new files. Cloud storage systems automatically create and retain versions of files and data as a safe guard. Whenever a user wants to revert some change, they can make a request to their Cloud storage service provider to have the change reverted. This process is streamlined, simple, and efficient.

3.2.5 File Sharing and Synchronization

Sharing files and data between authorized users is also easier through a Cloud storage system. Cloud storage systems have sophisticated authentication mechanisms which can grant read-only or read-and-write access to fellow users in the system if the original owner of a file allows it. The owner can grant these settings through the user interface of the Cloud storage system. Fellow users can then log onto the system and download a copy of the file whenever and from wherever they wish. If given write access, the files shared through a Cloud storage system can become a working repository where every change is always synchronized between all users. This form of file and data sharing is much more efficient in terms of storage space compared to emails, where the file would have to be replicated as many times as there are users, and where changes and updates must also be replicated in such a matter to have everyone on track.

3.2.6 A Way to Backup Data

Cloud storage allows users to easily apply the principle of keeping off-site backups of their data. By keeping a backup copy on the Cloud, any local catastrophes such as a complete equipment failure of the local hard drive will not affect or compromise the data stored on the Cloud. Users can often quickly recover their data, and get back up to speed with their work and tasks when such events occur with the assistance of Cloud storage.

3.3 Cloud Storage Costs

There are a series of costs associated with using Cloud storage. Generally speaking there are two segments of costs which are shown in on Figure 2.

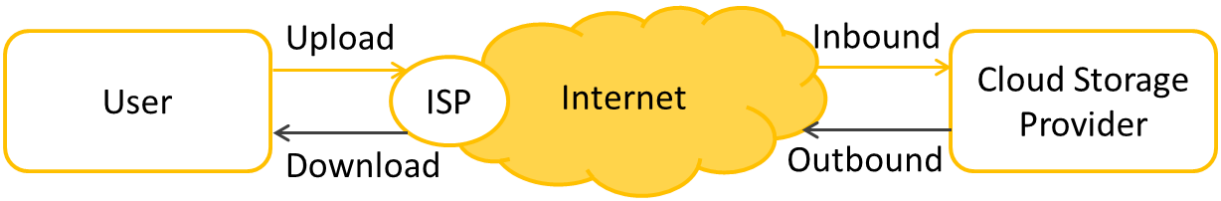

Figure 2 – Cloud Storage Cost Model

The two segments are the internet connection costs paid by the user to their internet service provider to connect to and use the internet, and Cloud storage provider costs paid by the user to specifically use the Cloud storage service. Internet connection costs include all of the data uploaded to, or downloaded from the internet. Cloud storage provider costs include potentially inbound traffic fees, outbound traffic fees, and storage fees. This section begins with a survey of internet connection and Cloud storage provider costs from the most popular providers in Canada and USA. Then, we present a survey of local disk storage prices to establish a cost reference point, to frame the discussions off the value and economic effects of pricing in Cloud storage.

3.3.1 Internet Connection Costs

In North America, most home internet service providers offer different plans where the main characteristic difference is the download and upload speeds. Generally, the download speeds offered are much higher than the upload speeds. In Canada during the year of 2014, home and business internet plans also have data usage allowance limits, where as in USA there isn't. The plans and data rates offered by some of the most popular service providers in Canada and USA as of February 2014 are shown in Table 4.

Table 4 – Internet Service Provider Pricing [35] [36] [37] [38] [39] [40] [41]

	Monthly Price	Download Speed (Mbps)	Upload Speed (Mbps)	Usage Allowance (GB/month)	Modem Surcharge	Usage Based Billing beyond Allowance	Price per GB	Price per Mbps (Download)	Price per Mbps (Upload)
Rogers (Canada; Cable)	$225.99	350	350	2000	No	N/A	$0.11	$0.65	$0.65
	$125.99	250	20	1000	Yes	$1/GB	$0.13	$0.50	$6.30
	$77.99	45	4	150	Yes	$2/GB	$0.52	$1.73	$19.50
	$67.99	35	3	120	Yes	$2/GB	$0.57	$1.94	$22.66
	$54.99	25	2	80	Yes	$2/GB	$0.69	$2.20	$27.50
	$44.49	6	0.25	20	Yes	N/A	$2.22	$7.42	$177.96
Bell (Canada; Fibre)	$152.95	175	175	300	No	$2/GB	$0.51	$0.87	$0.87
	$85.95	50	10	175	No	$2/GB	$0.49	$1.72	$8.60
	$60.95	25	10	100	No	$2/GB	$0.61	$2.44	$6.10
	$52.95	15	10	60	No	$2/GB	$0.88	$3.53	$5.30
	$42.95	5	1	20	No	$4/GB	$2.15	$8.59	$42.95
TekSavvy (Canada; DSL/Cable)	$86.95	150	10	300	Yes	$0.50/GB	$0.29	$0.58	$8.70
	$54.99	50	10	300	Yes	$0.25/GB	$0.18	$1.10	$5.50
	$56.95	45	4	300	Yes	$0.50/GB	$0.19	$1.27	$14.24
	$51.95	35	3	300	Yes	$0.50/GB	$0.17	$1.48	$17.32
	$39.99	25	10	300	Yes	$0.25/GB	$0.13	$1.60	$4.00
	$34.99	15	10	300	Yes	$0.25/GB	$0.12	$2.33	$3.50
	$29.99	7	1	300	Yes	$0.25/GB	$0.10	$4.28	$29.99
Shaw (Canada; Cable)	$120.00	250	15	1000	No	N/A	$0.12	$0.48	$8.00
	$90.00	100	5	500	No	N/A	$0.18	$0.90	$18.00
	$80.00	50	3	400	No	N/A	$0.20	$1.60	$26.67
	$60.00	25	2.5	250	No	N/A	$0.24	$2.40	$24.00
	$55.00	10	0.5	125	No	N/A	$0.44	$5.50	$110.00
AT&T (USA Cable)	$71.00	24	3	250	Yes	$0.2/GB	$0.28	$2.96	$23.67
	$61.00	18	1.5	250	Yes	$0.2/GB	$0.24	$3.39	$40.67
	$56.00	12	1.5	250	Yes	$0.2/GB	$0.22	$4.67	$37.33
	$51.00	6	1	250	Yes	$0.2/GB	$0.20	$8.50	$51.00
	$46.00	3	1	250	Yes	$0.2/GB	$0.18	$15.33	$46.00
Verizon (USA; Fibre)	$299.99	500	100	Unlimited	Yes	N/A	N/A	$0.60	$3.00
	$209.99	300	65	Unlimited	Yes	N/A	N/A	$0.70	$3.23
	$129.99	150	65	Unlimited	Yes	N/A	N/A	$0.87	$2.00
	$89.99	75	35	Unlimited	Yes	N/A	N/A	$1.20	$2.57
	$79.99	50	25	Unlimited	Yes	N/A	N/A	$1.60	$3.20
	$69.99	15	5	Unlimited	Yes	N/A	N/A	$4.67	$14.00
Comcast (USA; Cable)	$114.95	105	30	Unlimited	Yes	N/A	N/A	$1.09	$3.83
	$76.95	50	15	Unlimited	Yes	N/A	N/A	$1.54	$5.13
	$64.95	25	8	Unlimited	Yes	N/A	N/A	$2.60	$8.12
	$49.95	6	1.5	Unlimited	Yes	N/A	N/A	$8.33	$33.30

If a company puts a cap on usage allowance it is easy to calculate the precise cost for every unit of data transferred. The computed values of cost per GB are shown in

Table 4, showing price ranging from $0.10/GB to $2.22/GB. In addition, the usage based billing prices for data used beyond the allowance is anywhere from $0.20/GB to $4.00/GB. Generally a more expensive plan includes a higher usage allowance, at a cheaper price per GB of data transferred. Companies tabulate both download and upload activities to calculate how much data has been used within the allowance. A user would regard this as a fixed cost per GB of data sent to or retrieved from the Cloud.

In USA where most companies do not place a limit on usage, it is much easier to compare the time cost of transferring data to and from the Cloud. The higher the transfer speed, the less time it takes to transfer the data. This is beneficial for users since they can then spend the time on other activities, or free the bandwidth for other internet uses. The price for each unit of transfer speed could be used to compare the different plans. The computed values of cost per unit of transfer speed ($/Mbps) are shown as well in Table 4. Generally, a more expensive plan provides faster download and upload speeds and a cheaper unit price for every incremental unit of speed.

It would seem that no matter which of the two units of measure – price per data transferred or price per unit of transfer speed – the expensive plans are favored for their increased limits and lower unit costs. In practice however, each user would often choose an internet service plan based upon their available house hold budget and usage requirements. The use of Cloud storage services certainly increases the need for higher usage allowance.

Generally, these costs and parameters are not choices which the Cloud storage software and system can make, but has to work with to accomplish its goals. A Cloud storage system would be most optimal at minimizing the amount of data transfer necessary to accomplish each task. By minimizing the amount of data transferred, the system can minimize the data costs if usage allowance limits are present, and also accomplish transfers faster, minimizing the total time necessary to accomplish each task. No matter what a user chooses as their internet service plan, the costs can be minimized.

3.3.2 Cloud Storage Provider Costs

Cloud storage service providers often operate on a "freemium"-style business model, where an initial amount of storage is free for each user, and subsequent amounts of storage service is given in exchange for monetary gains. The most popular Cloud storage providers in North America are Dropbox [42], Amazon S3 [43], Microsoft SkyDrive [44], Apple iCloud [45], and Google Drive [46] in no particular order. We consider only the paid storage services for comparison since it is desirable to attribute a finite cost figure. It is worth noting that each of these five storage providers have free services too subject to space quotas or other limitations, shown in Table 5.

Table 5 – Free Tier Data Storage Limits for Cloud Providers [42] [43] [44] [45] [46]

	Dropbox	Amazon S3	Microsoft OneDrive	Apple	Google Drive
Free Storage Amount (GB)	2	5	7	5	15

Table 6 and Table 7 show the storage costs for February 2014 and February 2013, respectively, as listed by each of the provider's websites for various tiers of storage. Between February 2013 and February 2014, Microsoft has rebranded its SkyDrive

service to OneDrive. The pricing has also changed for these two services, overall increasing its prices in 2014. The 25GB data tier has also been eliminated. Overall costs range from Google Drive's $0.050/GB/month on the low end to Apple iCloud's $0.167/GB/month on the high end.

Table 6 – Cloud Storage Costs, February 2014 [42] [43] [44] [45] [46]

2014	Monthly Price ($USD)					Effective Price per GB per Month ($USD)				
Storage (GB)	Dropbox	Amazon S3	Microsoft OneDrive	Apple iCloud	Google Drive	Dropbox	Amazon S3	Microsoft OneDrive	Apple iCloud	Google Drive
10		0.85		1.67			0.085		0.167	
20		1.70		3.33			0.085		0.167	
50		4.25	5.49	8.33			0.085	0.110	0.167	
100	9.99	8.50	8.49		4.99	0.100	0.085	0.085		0.050
200	19.99	17.00	12.49		9.99	0.100	0.085	0.062		0.050
400		34.00			19.99		0.085			0.050
500	49.99	42.50				0.100	0.085			
1000		85.00			49.99		0.085			0.050
2000		160.00			99.99		0.080			
4000		310.00			199.99		0.078			
8000		610.00			399.99		0.076			
16000		1210.00			799.99		0.076			
50000		3760.00					0.075			
500000		30760.00					0.062			

Table 7 – Cloud Storage Costs, February 2013 [42] [43] [44] [45] [46]

2013	Monthly Price ($USD)					Effective Price per GB per Month ($USD)				
Storage (GB)	Dropbox	Amazon S3	Microsoft OneDrive	Apple iCloud	Google Drive	Dropbox	Amazon S3	Microsoft OneDrive	Apple iCloud	Google Drive
10		0.82		1.67			0.082		0.167	
20		1.62	0.83	3.33			0.081	0.042	0.167	
25		2.02			2.49		0.081			0.100
50		4.02	2.08	8.33			0.080	0.042	0.167	
100	9.99	7.52	4.17		4.99	0.100	0.075	0.042		0.050
200	19.99	14.52			9.99	0.100	0.073			0.050
400		28.52			19.99		0.071			0.050
500	49.99	35.52				0.100	0.071			
1000		68.02			49.99		0.068			0.050

Of the five Cloud service providers, only Amazon S3 charges a transmission cost for data. It charges a transmission fee for downloading data off of its servers. It does not

charge transmission fee for uploading any data to its servers. The prices for transmitting different amounts of data are shown in Table 8.

Table 8 – Transmission Costs for Amazon S3, February 2014 [43]

2014	Price ($USD)	Effective Price per GB ($USD)
Transmission (GB)	Amazon S3 Outbound	S3 Outbound
10	1.08	0.108
20	2.28	0.114
50	5.88	0.118
100	11.88	0.119
150	17.88	0.119
200	23.88	0.119
400	47.88	0.120
500	59.88	0.120
1000	119.88	0.120
10000	1199.88	0.120
50000	4799.88	0.096
150000	11799.88	0.079
500000	29299.88	0.059

Overall, Amazon S3 charges on average $0.118/GB outgoing for any amount of data up to 10TB, reducing the rate in subsequent service tiers to as low as $0.059/GB.

Any consumer side Cloud storage software systems should take into effect both the difference in prices and features of these providers, as well as the dynamic nature of the market place.

3.3.3 Comparison with Local Disk Storage Costs

Section 3.3.2 showed that Cloud storage providers charge between $0.050/GB/month to $0.167/GB/month for storing a user's files on their Clouds. It is useful to know the range of unit costs for local disk storage as well, to see relatively how much more users would have to pay to take advantage of the benefits of Cloud storage. Table 9 shows a sample of hard drive prices indexed in January, 2014 by a well-known product feature comparison and price ranking website called PCPartPicker [47].

Table 9 – Local Disk Storage Costs, January 2014 [47]

Model	Type	Form	RPM	Capacity (TB)	Cache (MB)	Price ($)	Price / GB ($)
Western Digital WD2500AAKX	HDD	3.5"	7200	250	16	57	0.2280
Seagate ST9250610NS		3.5"	7200	250	64	126	0.5040
Seagate ST500DM002		3.5"	7200	500	16	52.99	0.1060
Seagate ST9500622NS		**3.5"**	**7200**	**500**	**64**	**416.99**	**0.8340**
Seagate ST1000DM003		3.5"	7200	1000	64	59.99	0.0600
Seagate ST91000640SS		3.5"	7200	1000	64	258.38	0.2584
Seagate ST2000DM001		3.5"	7200	2000	64	89.79	0.0449
Seagate ST32000644NS		3.5"	7200	2000	64	279.98	0.1400
Seagate ST3000DM001		**3.5"**	**7200**	**3000**	**64**	**109.99**	**0.0367**
Seagate ST33000651NS		3.5"	7200	3000	64	475.86	0.1586
Seagate ST4000VN000		3.5"	7200	4000	64	189.95	0.0475
Hitachi 0B26885		3.5"	7200	4000	64	666.68	0.1667
Hitachi 0F18335		3.5"	7200	6000	64	969.47	0.1616
PNY SSD9SC480GMDA-RB	SSD	2.5"	N/A	480	N/A	167.75	0.3495
OCZ OCT1-25SAT3-1T		2.5"	N/A	1000	N/A	2071.56	2.0716
Seagate ST4000DX001	Hybrid	3.5"	N/A	4000	64	200.98	0.0502
Seagate STBD1000400		3.5"	N/A	1000	64	129.75	0.1298

For standard magnetic disk hard disk drives (HDD), the unit price ranges from as low as $0.0367/GB to as high as $0.8340/GB. On the low end of the price range, storing data on the Cloud seems to be much more expensive. For the first month, storing data on the Cloud would cost $0.050/GB, while buying a hard drive can cost $0.0367/GB. After the first month, the Cloud storage costs continue to add but the hard drive would have been paid for already. On the high end of the price ranges, buying a hard disk would cost $0.8340/GB while the first month Cloud storage cost would be $0.167/GB. Users can reap roughly 5 months of Cloud storage benefits before the costs would break even. However, it is naive to assume users would purchase hard disks at the high end of the price range. Likewise, Cloud storage providers aim to maximize profits so it is unlikely they would purchase hard drives on the high end of the price range.

For specialty disk drives such as Solid State Drives (SSD) and Hybrid Disk Drives, Table 9 shows their lowest and highest unit costs. They range from $0.3495/GB to

$2.0716/GB for SSD drives, and $0.0502/GB to $0.1298/GB for Hybrid drives. Users gain significant disk throughput and performance with these specialty drives, but their unit costs are much higher than HDDs. Specialty disk drives would be used in high performance computation Clouds, however a Cloud storage service provider would be unlikely to invest in using specialty disk drives to store users' files since they want to maximize revenue and profits. Start-up companies and small to medium size businesses will likely pay to store files on the Cloud because they can save management and operation costs as mentioned in Section 3.2.1.

Overall, we can conclude that users can achieve lower short term and long term storage costs by simply purchasing hard drives from a computer store and placing their data into those hard drives. To enjoy the benefits of Cloud storage, users must pay a non-trivial fee once their use exceeds the free storage quotas. Thus, it is important for any Cloud storage system to minimize the short term and long term storage costs.

3.3.4 Economic Effects

The data in Section 3.3.2 shows an interesting economic model for Amazon's S3 service, whereby it is free to upload data to Amazon; however it is not free to download the same data from Amazon. This business model favors a different type of application such as data-mining where users upload a lot of data to the Cloud, then perform extensive computation on the data obtain specific results that has a substantially smaller file size than the data, and then download only the results.

To generalize these business models and practices, it is best to consider that a Cloud storage provider can charge for both incoming and outgoing data, often charges a fee

for data stored on their Clouds, can impose transfer and storage limits, and can change the prices and rules for all of these at any time.

Consider initially two Cloud storage providers who both charge for the same transmission and storage costs for users. The user chooses one of the two providers and puts all of their data on to that Cloud. The costs paid so far is a one-time fee for transmission (this includes fees charged by the storage service provider and those charged by internet service providers), and an ongoing fee for storage. Consider further now that the other Cloud storage provider decides to reduce their storage fees as an incentive to attract business. The user is now faced with a dilemma of choosing to stay with their existing provider but pay higher long term costs, or pay an expensive transmission fee now (one time to download the data, and one time to upload the data) and move the data to the other provider in order to take advantage of the long term savings. Total transmission fees are typically higher than storage fees, as shown in Sections 3.3.1 and 3.3.2. This creates an economic barrier and condition which Abu-Libdeh et al. calls Storage Vendor Lock-in [3], where once a user commits to storing data in one provider they are no longer economically able to afford to move the data to a different provider without paying a hefty fee to move the data out.

Even if the monetary price was free for all transmissions and storage, there is still a resource consumption problem with this method of moving between data storage providers. Considering that bandwidth is often a critical resource bottleneck for modern day Clouds and networks, it is ideal to reduce the bandwidth consumed for this task. A direct transfer between the providers can reduce overall resource consumption, however currently it is unforeseeable that there would be direct

connections and communications between Cloud storage providers due to business and competition. From an energy perspective, direct transfers would save electricity as well compared to downloading then re-uploading.

A Cloud storage subsystem would have to keep in mind the market dynamics and work actively to prevent data lock in, while attempting to minimize a user's costs for storing data on a Cloud.

3.4 Chapter Summary

This chapter presented the risks, benefits, and costs of using Cloud storage. For files that are stored on the Cloud, they are at risk from malicious attacks from anywhere around the world, and at risk of being lost or corrupted due to service provider mistakes and equipment failures. Users must implicitly trust and depend upon the service provider to safeguard their files, and to provide uninterrupted access to the files. Difference of interpretation of the notion of data privacy, as well as potentially conflicting laws around the world also play havoc to the Cloud storage ecosystem. Users may inadvertently have their privacy compromised if they place their files in the wrong geographical locations.

Users can also benefit from placing files on the Cloud, including potential cost savings for start-up companies, improved global resource utilization, having world wide access to their files, having file versioning and recovery services, having data sharing services in team settings, and having an easy way to apply the principle of off-site data protection.

We also presented a survey of the most popular internet service providers' plans and rates in Canada and USA, as well as the costs charged by the most popular Cloud

storage providers for data stored beyond the free storage quota to show the financial costs a user encounters when using Cloud storage. We compared the costs to using local disk storage, and also discussed the economic effect of Storage Vendor Lock-in where users cannot move their data between service providers without having to pay a heavy fee.

At the intersection of the risks, benefits, and costs, we can see that users must juggle a lot of factors to optimally take advantage of Cloud storage. The next chapter concretely defines this problem.

Chapter 4: Cloud Storage Problem Definition

From the analysis presented in Chapters 2 and 3, we can see that users have very little control over their files and data stored on the Cloud, and are at the whims of Cloud storage providers in terms of service availability, data security, data privacy, and pricing. At the same time, there are numerous advantages to storing data on the Cloud, such as ease of access, off site back-ups, potential cost savings, and improved resource utilization. Users currently must make a conscious choice of accepting and facing these drawbacks in order to reap the benefits of storing data on the Cloud. Tackling this trade-off is the central problem for Cloud storage.

We want to design a sound solution from a user's perspective to resolve this trade-off in such a way that minimizes or eliminates as much of the problems as possible, while maintaining or enhancing the benefits of using Cloud storage. An ideal solution would eliminate all of the problems and enhance the benefits, so that users would no longer have to make this trade-off when they want to store files on the Cloud.

To summarize the analysis, the risks and problems with using Cloud storage are:

1. Malicious attacks can come from anywhere around the world, often targeting the authentication mechanism of Cloud storage providers
2. Weak access control mechanisms may allow users to see other users' data
3. Routine maintenance or outages can cause inconveniences and delays

4. Data loss and corruption risk is always present due to equipment failure and potential natural disasters
5. Implicit trust of storage providers necessary, but problematic as the control of the data resides on the service provider, while the responsibility for the data rests on the user
6. Physical location of the Cloud data is unknown to the user, and users have no control over the physical placement of their data
7. Economic and business models of Cloud providers often create problems of data lock-in and an inability for users to move data from one provider to another without paying high fees
8. Changing pricing between Cloud providers is hard to track for users, and hard to optimize for a least-cost strategy since data move is currently necessary to take advantage of lower prices
9. Requirements of local and international law may inadvertently violate a person's reasonable expectation of data privacy

The benefits of utilizing Cloud storage are:

1. Pay only for the amount of data storage used
2. Flexible storage quotas, adjusted on-demand
3. Lower upfront costs, highly beneficial for start-up companies
4. Reduced management and maintenance fees
5. For short term projects, no need to worry about reselling hardware
6. Improved resource utilization, saving energy for the world
7. Access data from anywhere around the world
8. Automatic storage of data backups and revisions

9. Ease platform for data sharing between users
10. Changes are synchronized between a team when data is modified by team members
11. An application of off-site data protection principle, allowing users to recover data easily

Of the nine problems and risks outlined, the thesis will address all of them with the exception of problem 9. As previously discussed in Section 3.1.5 the solution to legal requirements necessitates political dialogue in both domestic and international settings to develop new standards and interpretations of the meaning of data privacy as well as tools for enforcement and compliance in an every increasingly online and connected world.

The next chapter presents the approach and methodology for solving the trade-off problem, along with critical analysis of the approach itself.

Chapter 5: Approach and Methodology

A promising solution has been proposed by Abu-Libdeh et al. in their paper "RACS: A Case for Cloud Storage Diversity" [3]. In the paper, they present an approach to the trade-off problem by applying erasure code algorithms to split a user's files into numerous pieces, add redundancy to these pieces to tolerate losses, and then send the file pieces to different Cloud providers. The focus of their paper was to resolve the economic issues of Cloud storage.

We adopt this approach, and propose a more comprehensive system design Framework to resolve not only the economic issues but also the security, reliability, and privacy issues of storing data on the Cloud. The Framework provides a sound template design for a practical storage software system, which users can run on their computers and mobile devices to reap the benefits of using Cloud storage without having to worry about the problems mentioned Chapter 4.

This chapter focuses on analyzing the approach, while Chapter 6 presents the Framework and detailed analysis of its components. This chapter begins by defining a formal model of erasure code file transformations in Section 5.1 to explain how erasure codes work in general. We then analyze Abu-Libdeh et al.'s approach and research work in Section 5.2 where we will mention the weaknesses of the approach. We explain a critical security problem which their work did not consider in Section 5.3. Section 5.4 shows how such an approach can solve the trade-off problem, and Section 5.5 examines how the approach can augment and enhance the benefits of

using Cloud storage. Lastly Section 5.6 compares the approach to the current best practice from a user's point of view.

5.1 Formal Model of Erasure Code File Transformations

Traditionally, erasure codes are used to add redundancy to small pieces of data, and then capture them in some form of a digital "container". The container could be a single file, a set of files, a single network packet, or multiple packets, etc. This thesis focuses on using a set of equal size files as the container. Erasure codes add redundancy to the data in the encoding transformation process, and reconstruct the data in the decoding transformation process. Both processes are mathematically related. Only a subset of the file pieces are used since redundancy was added during the encoding transformation.

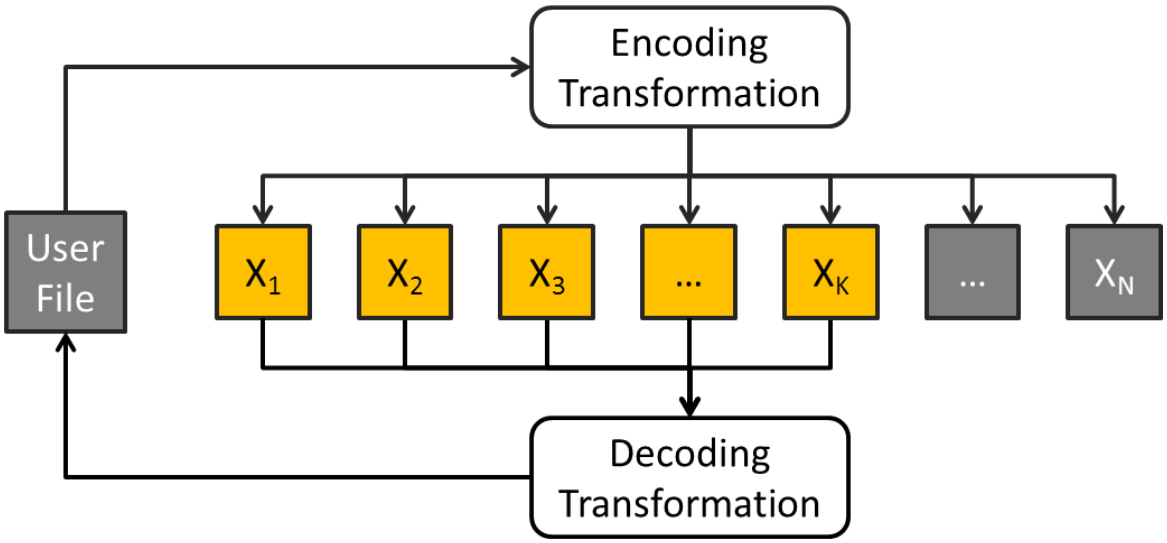

Figure 3 – Erasure Code File Transformation Model

There are many erasure codes in literature and in practice, each of them employing either different mathematical principles, or different configurations of the same principles [48] [49] [50] [51] [52] [53]. A general notation of (N, K) is used within

literature to denote the number of file pieces (N) generated in each encoding transformation, as well as the number of pieces (K) needed to reconstruct the original file in the decoding transformation. Generally, K is less than N to allow for (N – K) redundancy pieces. This is visualized in Figure 3, where a user's file is encoded into N file pieces denoted as X_1, X_2 … X_K … X_N. Some codes must use a specific labeled subset of K pieces for the decoding transformation, while other codes can use any subset of K pieces. Generally speaking, the use of an erasure code involves more than simply implementing the transformation algorithm as proper management of the file pieces is necessary before any decoding transformations can occur. This often involves labeling all of the file pieces and capturing this information in a metadata file.

Literature refers this type of mathematical transformation algorithms by various names in the domains of network coding, cryptographic systems, and storage systems. They have been called (N, K) channel codes, (N, K) error-correcting codes, (N, K)-threshold schemes, distributed key systems, secret sharing systems, and so on. The varied names are given to facilitate discussions with a focus in their respective domains. Erasure Code is another name for the same thing, commonly used within network coding and storage systems domains. The notion of an Erasure Code implies both the mathematical and computational algorithm used by the code, as well as the procedural use of the code. Erasure Codes must be configured for specific, valid, (N, K) pair values to be used. Whenever literature directly refers to an erasure code by name, they often imply a focused discussion on the algorithm aspect. When they refer to the code by a specific (N, K) configuration, they often imply a focused discussion on that configuration. This thesis focuses on both aspects of an Erasure Code, but they are discussed separately.

5.2 Redundant Array of Cloud Storage System

Abu-Libdeh et al. [3] of Cornell University approached the problem from an economics and data loss tolerance point of view. They developed a software system called Redundant Array of Cloud Storage (RACS) which uses the Reed-Solomon code [53] as its erasure code. Their efforts aim to give users the ability to tolerate service outages and data loss, adapt to ongoing price changes and provider availability in the marketplace, and control total storage costs. To the best of the author's knowledge, this is the first system which applies the approach specifically on Cloud storage problems, which made it interesting as a new application domain.

Abu-Libdeh et al. discussed two economic benefits and one security benefit which this approach can offer, namely that it can help users avoid vendor lock-in, reduce the cost of switching service providers, and tolerate provider outages and failures. As validation and proof, their studies included a cost estimation and trace driven simulation of moving all of the data contained in the Internet Archive website to a new storage provider. Their simulation results showed up to 80% cost savings for service provider migration tasks. However, for normal uploads and downloads of files it showed an average increase of 50% in costs, corresponding to the efficiencies of their chosen Reed-Solomon code configurations. They did not specifically mention the reasons for choosing Reed-Solomon as the erasure code.

A feature of RACS is its ability to operate through multiple running instances of the program, in parallel, within a server environment using Apache ZooKeeper as a distributed synchronization system between each instance. In this set up, RACS can service multiple users at the same time and avoid performance bottlenecks caused by

each instance since user requests can be serviced by another available instance. The architecture of RACS is shown in Figure 4.

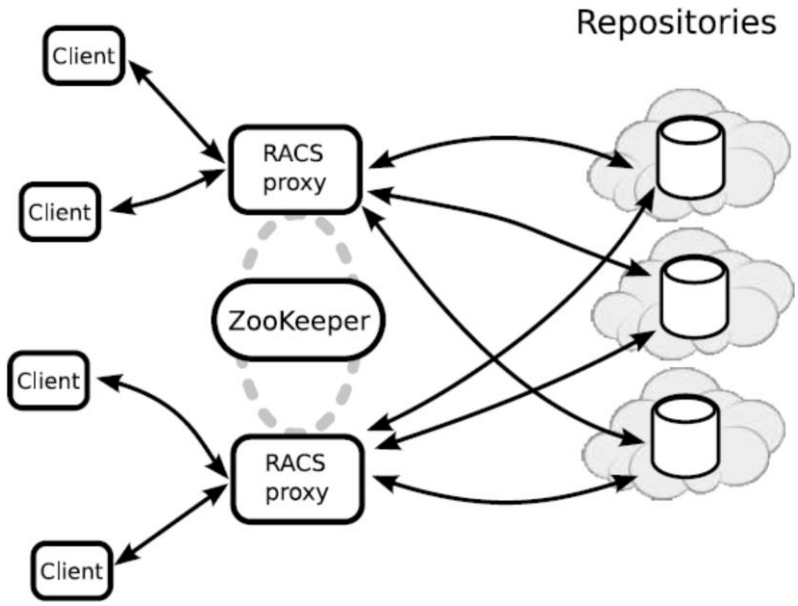

Figure 4 – Redundant Array of Cloud Storage Architecture [3]

Each RACS proxy instance contains a functional file transformation system, along with the management systems and communications systems to send the file pieces to various Cloud storage providers. Their system was implemented with a focus on being used on the Amazon S3 storage system. They wrote Repository Adaptors, or simply software APIs, which allow their system to work with other storage providers using a uniform interface.

The paper mentions the use of "policy hints" to capture user preferences of which storage providers to use. It also mentions a key point to combine user preference with the need to load balance between the storage providers in order to achieve optimal economic freedom and data security.

A software performance analysis was also carried out by Abu-Libdeh et al. showing encoding throughput of 95MB/s and decoding throughput of 151MB/s using an open source erasure coding library called Zfec on a 2GHz Intel Core 2 Duo powered computer. Zfec includes an implementation of the Reed-Solomon Code. Relative to end users, these throughput rates are fast since the expected internet throughputs are much slower. As a software system installed on end users' computers, the performance bottleneck would be on their internet connection. In corporate settings where high speed and high performance networks might be common, the throughputs might cause delays if the network supports 1 Gigabit/s or higher bandwidths.

There were three main weaknesses of the approach mentioned in the paper. First, the total storage space used increases by a factor of N ÷ K, which results in higher transmission costs and storage costs. Second, the number of requests issued to the Cloud increases by a factor of N since every file piece is treated and considered to be a file by the storage provider. Operations such as creating and deleting files must wait for all requests to complete. Lastly, the system introduces latency as all files must undergo encoding and decoding transformations.

This thesis makes a number of contributions and improvements to the work done by Abu-Libdeh et al. First, we examined in much more depth the security issues and economic problems of using Cloud storage in Chapters 2 and 3. In Sections 5.4 and 5.5, we will show how to resolve all of these problems using the approach. Second, the research work from Abu-Libdeh et al. does not state how they manage their metadata on the files, for which this thesis proposes a solution in the form of using a peer-to-peer network to replicate the metadata across a user's devices. Third, their

research work does not state why they specifically chose Reed-Solomon coding as the erasure code. This thesis examines families and classes of erasure codes in Chapter 6 to study the security properties and performance of different erasure codes, as well as their applicability towards Cloud storage problems. Lastly, we contrast this approach to traditional remote storage paradigms in Chapter 7.

5.3 The Dictionary Attack Problem

Encoding files into file pieces can still present a cryptographic security problem, one which was not discussed by Abu-Libdeh et al. Given a sufficient number of file pieces less than the threshold K and external knowledge about the data, one could plausibly guess at the data of any missing pieces with very high accuracy. Any missing pieces required to meet the threshold could be deduced without having to obtain them from the Cloud. This is called the dictionary attack in security research. Consider the following example:

Let the original file contain the word **PASSWORD**, and consider that it is split into four pieces consisting of **PA**, **SS**, **WO**, and **RD**. If an attacker obtains any 3 of these blocks – for example **__SSWORD**, **PA__WORD**, **PASS__RD**, or **PASSWO__** – the remaining block can be guessed quite easily using an English language dictionary search. In this case, the attacker has external knowledge about the data, namely that the contents are English words and that in total there are 8 characters. If an attacker obtains any 2 of the blocks – for example **____WORD**, **__SS__RD**, or **__SSWO__** – guessing becomes more difficult. Some plausible English words fitting **____WORD** might be **BUZZWORD** or **FOREWORD**. If the attacker did not know how many characters there were in total, then words such as **CROSSWORD**, **SWORD**, and

AFTERWORD are also plausible guesses. Having more plausible results increases the chances of the attacker failing to obtain the true original data.

Some erasure codes will encode a file in such a way that none of the file pieces contain, directly, the text or data in the original file. For the above example, these types of erasure codes would encode **PASSWORD** to another set of characters, such as **WR**, **SL**, **AB**, and **EK**. These codes have the Confusion security property, while others such as Reed-Solomon Codes do not have the property.

5.4 Addressing Secure Cloud Storage Problems

By splitting and spreading out the pieces of the file to multiple Cloud storage providers (at least two providers), the problems outlined in Chapter 4 can be addressed. We examine each problem one at a time:

1. **Malicious attacks can come from anywhere around the world, often targeting the authentication mechanism of Cloud storage providers**

An attacker must obtain at least K of the N pieces of files in order to accomplish his or her objective of obtaining the original file, presuming that this original file contains highly valuable and sensitive information to both the user and the attacker. If the N pieces are spread out in such a way that we ensure no single Cloud provider has K pieces, then the attacker must compromise at least as many Cloud providers at the same time as it takes to obtain the K pieces necessary. By convention, this is at least two providers. If the attacker were to attempt to break the authentication mechanisms of the providers at different times, then it becomes and ever increasingly difficult for every subsequent attack as companies and users will have the time to react to the first attack to further safeguard their systems and data.

Further, it can also be presumed that the attack against one provider will not work against another provider without significant adaptation. While there are common security best practices used by all companies, the specific security architecture of each provider is different, which necessitates a different way of attacking that provider. While it is theoretically impossible to prevent simultaneous attacks, nor the complete elimination of this risk, the Framework will further minimize the risk of malicious attacks as it increases the amount of work necessary for attackers before they can achieve their objectives, compared to the current practice of putting all of the data on a single Cloud storage provider.

2. Weak access control mechanisms may allow users to see other users' data

When a Cloud storage provider has a weak access control mechanism, the ultimate solution is for that provider to modify and improve their mechanism so that users cannot see each other's data. Since the Framework only allows less than K file pieces to be stored on each Cloud provider, other users would not be able to reconstruct the original file without obtaining the remaining pieces from another compromised Cloud storage provider. It is unlikely for any two Cloud storage providers to have the same access control weaknesses, as their architectures would have some differences to avoid legal copyright problems. Further, even if two Cloud storage providers would have such weaknesses, it is unlikely that they would have it at the same time.

3. Routine maintenance or outages can cause inconveniences and delays

System maintenances and temporary outages do occur with Cloud storage providers for many reasons. Sometimes their entire system must undergo an update at the same time, whereby they cannot schedule piecewise updates to their subsystems.

Sometimes it could occur due to electrical outages or other unforeseen events. The management plan and system architecture of the provider's service often have the most direct impact towards the frequency maintenance activities. Thus the direct solution to reducing outages and maintenance rests with better management plans and architectures that allow for piece-wise upgrades to their systems. When a maintenance or outage occurs for a specific Cloud provider, the Framework can reconstruct the pieces stored in that provider by downloading the necessary file pieces from the other Cloud providers. Erasure codes not only allow the reconstruction of the entire file, but also the pieces. Some of the code algorithms will require a complete reconstruction of the file followed by re-splitting the file into new pieces, while other codes can allow piece wise reconstruction using specific pieces. By reconstructing the pieces stored in the Cloud provider experiencing an outage, the user can continue to have access to the original file and data while the Cloud provider fixes their systems.

4. Data loss and corruption risk is always present due to equipment failure and potential natural disasters

Piecewise file reconstruction can also be used to safeguard against complete data loss or corruption due to equipment failure and damaging natural disasters. Complete data loss can be treated the same as a maintenance outage, whereby the system simply reconstructs the missing pieces. The system can compute the hash value of each file piece, and use that to check against the stored hash values in the metadata for those pieces to verify that the pieces have not been modified while being stored on the Cloud. If any checks fail, the corrupted pieces can be

reconstructed using the remaining file pieces as long as K total pieces or more are intact.

If less than K total pieces remain intact, then the system will not be able to reconstruct the file from the pieces, but if an original copy was retained by the user then it could reconstruct the pieces by splitting the original file again.

5. **Implicit trust of storage providers necessary, but problematic as the control of the data resides on the service provider, while the responsibility for the data rests on the user**

The Framework can treat the file and data to help it tolerate against attacks on a storage provider, against weaker system design where user's files might be viewed by other users, against system outages, and against total data loss and corruption of a single provider. Depending on the way the system distributes the specific file pieces, the files can tolerate problems present in multiple providers at the same time. For example, where one provider experiences accidentally deletes a user's file piece, while another provider experiences an equipment failure. However, as K file pieces must remain intact, some subset of the total number of providers must remain functioning. The implicit dependency and need to trust a single storage provider can effectively be eliminated; however the ecosystem of providers must still be trustworthy and generally reliable. Since the user can recover all data from the K file pieces, they now have the ultimate control over all of their data. If all providers are neither reliable nor trustworthy, the Framework would work better by sending the file pieces to multiple local storage mediums, such as USB flash drives, to safeguard

against equipment failure. In such a case, Cloud storage itself would have a systemic issue within the entire ecosystem.

6. Physical location of the Cloud data is unknown to the user, and users have no control over the physical placement of their data

The specific physical location of a user's data within a Cloud is very much undeterminable from a user's perspective. On the one hand, the lack of such knowledge safeguards users from any potential physical attacks of data centers, since attackers wouldn't know which data center to target to steal hard drives. On the other hand, most of the major Cloud storage providers are expanding worldwide with physical data centers in all major continents, and even choosing a provider does not necessarily imply choosing the continents, countries, or cities of where the data would be stored. However, numerous small Cloud storage providers exist, providing storage services tailored for specialized markets, for example CareCloud is a specific Cloud storage provider for health care data in USA, which claims to be HIPPA certified for US law requirements [54]. HIPPA specifically requires providers to track the physical movement and locations of any healthcare data within their systems [55], and to be able to audit and verify such movements [56].

The Framework takes into account the geographic locations of each Cloud storage provider, which can give users control of the physical placement of their data to some degree. Efforts to track and locate all of the physical locations of the data centers for each Cloud provider may be necessary in order to completely address this problem, however the efforts would be exhaustive if done by manual labour, and the

correctness of the report will deteriorate through time as companies expand or relocate their data centers.

7. Economic and business models of Cloud providers often create problems of data lock-in and an inability for users to move data from one provider to another without paying high fees

Recall the scenario discussed earlier in Section 3.3.4, where the user has their data in the more expensive of two Cloud storage service providers. The user is faced with a dilemma of either paying higher long term costs by staying with their current provider, or an expensive transfer fee (outbound and inbound fees from providers and download and upload fees from internet service provider) to move their files to the less expensive of the two Cloud storage providers for long term savings. The Framework lets the user reconstruct the file pieces stored on the expensive provider locally on their computer, and then upload these directly to the less expensive provider. The user doesn't have to pay a download fee or an outbound data fee in order to take advantage of the savings. Although an upload fee and inbound data fee is still present, the total cost is less. The system can compute the time period for which the new storage cost plus move cost is equal to the storage cost of the former storage provider. Intuitively, storing a file on the Cloud any time after this time period will result in cost savings compared to staying with the current provider. A move can be made if the file is expected to be stored in the Cloud longer than the computed time period. We present this in further detail in Section 6.5.1. The reduction of the total cost is in one sense an economic freedom which the user can take advantage of through the Framework.

8. **Changing pricing between Cloud providers is hard to track for users, and hard to optimize for a least-cost strategy since data move is currently necessary to take advantage of lower prices**

The Framework will automatically track Cloud storage and internet prices, thus alleviates the need for the users to track the price changes. With all the available pricing data, the system can intelligently formulate a dynamic least-cost strategy balancing pricing, storage, and security requirements. Data move in the form discussed here in point 7 is still necessary to take advantage of lower storage prices, but the system will intelligently decide when and where to move the files according to all of the requirements.

5.5 Preserving Cloud Storage Benefits

The Framework preserves all existing benefits of storing data on the Cloud, and also enhances some of the benefits in intuitive ways. The benefits outlined in Chapter 4 are examined point by point below to see how the system will preserve or enhance the benefits.

1. **Pay only for the amount of data storage used**

The Framework will still allow the user to pay only for the amount of data storage used on the Cloud. Although the system does compute extra file pieces for the advantages of redundancy and security, the location and placement of these pieces could be on or off the Cloud. It is a flexible option for the user as to how many file pieces should be placed on the Cloud, and as a result how much costs they can expect by placing the file pieces on the Cloud. For example, if we only put less than K file pieces on the Cloud where K pieces constitute the size of the original file, then

the system allows the user to save on costs by not having to put as much data on the Cloud compared to using Cloud storage in the traditional sense.

2. Flexible storage quotas, adjusted on-demand

Since the Framework aims to distribute the file pieces across a number of storage providers as a security principle, indirectly this allows the user to not need to request for higher storage quotas from each provider, and generally remain lower in costs. Of the surveyed storage providers shown in Section 3.3.2, only Amazon and Microsoft has lower effective price per GB stored as a user requests for a higher quota. The other providers charge the same unit price at any storage tier. All of these providers have a free storage tier, thus the system can optimize costs by distributing the file pieces in such a way that it wouldn't use more than the free storage quota of space from each provider until there is no more free space remaining. The system can intelligently take advantage of the free storage spaces available in the Cloud storage market.

3. Lower upfront costs, highly beneficial for start-up companies

With sufficient market research and indexing, the Framework will contain a wealth of knowledge of the pricing, free storage limits, security features, and legal restrictions set by each storage provider for the data they host on their Clouds. Within this data set, the system could be used as a recommendation system where it suggests to the user where to place their data, how much storage it needs, and the costs to expect. From the perspective of a start-up company, the system could be advantageous in keeping start-up costs low by intelligently finding additional free or low cost storage

tiers and providers with suitable requirements for the business as its data storage needs grow.

4. Reduced management and maintenance fees

The aim of the Framework is to be as intelligent and automated as possible in managing the distribution and reassembly of the required file pieces, thus very little management work or interaction would be needed from the user once some initial configuration work is completed. Like most other existing Cloud storage tools and solutions, automation and machine intelligence reduces the amount of management and maintenance work on the overall file system which allows users to save time and companies to save costs.

5. For short term projects, no need to worry about reselling hardware

The Framework continues to allow users and companies not have to worry about reselling storage hardware as all of those costs are bear by the Cloud storage provider. The implementation of the system can be accomplished by a wide range of software languages for various architectures and platforms. The user does not need to invest in any specialized hardware in order to use this Framework.

6. Improved resource utilization, saving energy for the world

One of the main reasons for users to pre-emptively purchase a very large storage capacity hard disk is to accommodate any unexpected or unknown future storage needs. For portable computers it also makes sense to increase the storage capacity available given that there is usually space only for one or two hard disks per laptop. For desktops or older computers, it makes sense to consolidate many smaller

capacity hard disks into one large one for the purpose of saving energy and reducing weight. The use of Cloud storage extends the capacity limit since most files could be offloaded onto the Cloud. In one sense, a storage system that intelligently capitalizes on this flexibility can even maximize the utilization of local storage by finding the files that could be offloaded onto the Cloud and moving them automatically. Not only can the user have a virtually unlimited storage space, but the local hard disks can become smaller in actual storage capacity, reducing the global demand for the scares resources needed to produce so many high capacity but underutilized hard disks. Likewise, sending more files, including the redundant file pieces to the Cloud allows a Cloud provider to maximize their storage disk utilization and increase revenue.

7. Access data from anywhere around the world

Since the Framework uses metadata to keep track of where the file pieces are and the metadata is replicated across the user's devices through a peer-to-peer file system, the user is guaranteed to have access to their data anywhere around the world as long as they have an internet connection.

8. Automatic storage of data backups and revisions

The Framework can work in conjunction with already available backup and revision capable Cloud storage systems to archive versions of a file. In such settings, the Framework will simply split the updated file into the same number of resultant file pieces as the original file, and name them accordingly such that the Cloud storage systems register the new file pieces as a revision of their corresponding old file pieces.

Some erasure code algorithms can work with incremental changes to an original file, by computing the respective changes to each of the resultant file pieces. Using only these algorithms, the system can upload the new file pieces to the storage providers and add a corresponding entry in the metadata to track the revision of the file. This approach restricts the list of applicable algorithms, but can be used across any number of Cloud storage providers as the revision capability is provided by the Framework instead of the Cloud providers, and this approach is likely the most efficient in terms of the use of storage space.

9. Ease platform for data sharing between users

Users who share the same file will need to exchange the metadata and use the same storage system. Since the metadata is shared through a peer-to-peer network, the authorized users' devices and computers can be added as peers to the file's P2P network in order to receive the metadata. When sharing for specific files or folders stops, the Framework disconnects the relevant peers from the P2P networks and gracefully handles the files.

10. Changes are synchronized between a team when data is modified by team members

The metadata can also be used to track changes, in such a way that the team members only need to download the K pieces necessary to reconstruct the new updated file instead of all pieces.

11. An application of off-site data protection principle, allowing users to recover data easily

The Framework further enhances a user's ability to recover data through the application of the off-site data protection principle. Not only will it allow the user to tolerate their own hardware or system failure, but also failures of individual Cloud storage providers. As long as K pieces remain intact anywhere around the world, the user can recover their files.

5.6 Current User Best Practices

From a user's point of view, the current best practice is to simply encrypt all their files before putting them on the Cloud. This is often advocated by many consumer websites and blogs [57] along with suggestions and promotions of specific encryption software. These blog posts serve as a good means to raise awareness of the problems of Cloud storage, stimulate community discussions, and generally educating consumers on the ideas of encryption. However, there are a number of drawbacks to this methodology. The user must learn about encryption systems to properly apply encryption to their files, or implicitly trust the encryption software. Encrypting files involve selecting an appropriate encryption algorithm and system, determining the level of security needed and select a proper length of an encryption key corresponding to the level, and then apply the system in the proper procedure to encrypt the file. The user must also manage the encryption keys, and trust whomever they share the keys with if they want to share the files. If a user loses his or her encryption key, there is a likely chance that they will never be able to decrypt their files. Even if these steps were taken, it does not guarantee that an attacker won't obtain their data. If an attacker obtains a complete copy of the encrypted file, they may expend as much computational resources and time as needed to decrypt the file through brute force or other techniques.

The erasure code approach solves this problem by spreading the pieces among many Cloud providers, forcing an attacker to execute coordinated and concurrent attacks to multiple providers. This increases significantly the difficulty and amount of work necessary for an attacker to gain access to the data.

5.7 Chapter Summary

This chapter presented the approach to solve the Cloud storage problem which applies erasure code algorithms to split a user's files into numerous pieces, add redundancy to these pieces to tolerate losses, and then send the file pieces to different Cloud providers. The approach is an adaptation of the approach used by Abu-Libdeh et al. in their RACS system. We presented a formal model of erasure code file transformations to explain how erasure codes work at a high level, then we analyzed Abu-Libdeh et al.'s research work. We showed how their work was missing critical analysis in terms of data security by presenting the Dictionary Attack Problem, which shows a critical vulnerability in their system as attackers can guess missing file pieces given that they obtain a sufficient amount of data pieces.

We then addressed point by point how the approach itself can be used to resolve the risks and challenges of using Cloud storage, whilst enhancing the benefits. We also analyzed the current best practices for users, showing how despite the benefits of increasing awareness and educating the population about the risks of Cloud storage, the best practices still has vulnerabilities.

In the next chapter, we apply the approach by presenting the design and in-depth analysis of a system Framework that resolves the Cloud storage problem.

CHAPTER 6: CLOUD STORAGE FRAMEWORK

Designing a secure storage software Framework requires a thorough analysis of each of the components of the Framework. This chapter begins with a presentation and discussion of the components of the Framework in Section 6.1. The most critical component is the erasure code transformation system which carries out the encoding and decoding operations on files. The analysis metrics for erasure codes are presented in Section 6.2, followed by an in depth analysis of seven erasure codes in Section 6.3. The other components are described in Sections 6.4 and 6.5 respectively, followed by the chapter summary in Section 6.6.

6.1 GENERAL MODEL OF FRAMEWORK

A Cloud storage Framework is a template software system which takes some user files(s) as input, and transforms them into a proper set of file pieces as output to be stored on the Cloud, in such a way that some redundancy is added to afford data loss, corruption, service outage, or equipment failure. A high level representation is shown in Figure 5. The Framework has four high level systems. The erasure code transformation system encodes and decodes user files into file pieces (X_1, X_2, X_3 … X_N).

The Cloud storage management system selects Cloud providers and manages the upload and download of the file pieces to a number of Cloud providers. The choice of Cloud storage providers depend on their price, availability, geography, security, and other metrics. The choice is independent of the encoding and decoding transformations. Thus the system is able to tailor the choices of storage providers

towards any number of requirements, whether it is for law compliance, cost reduction, data security, or a combination thereof.

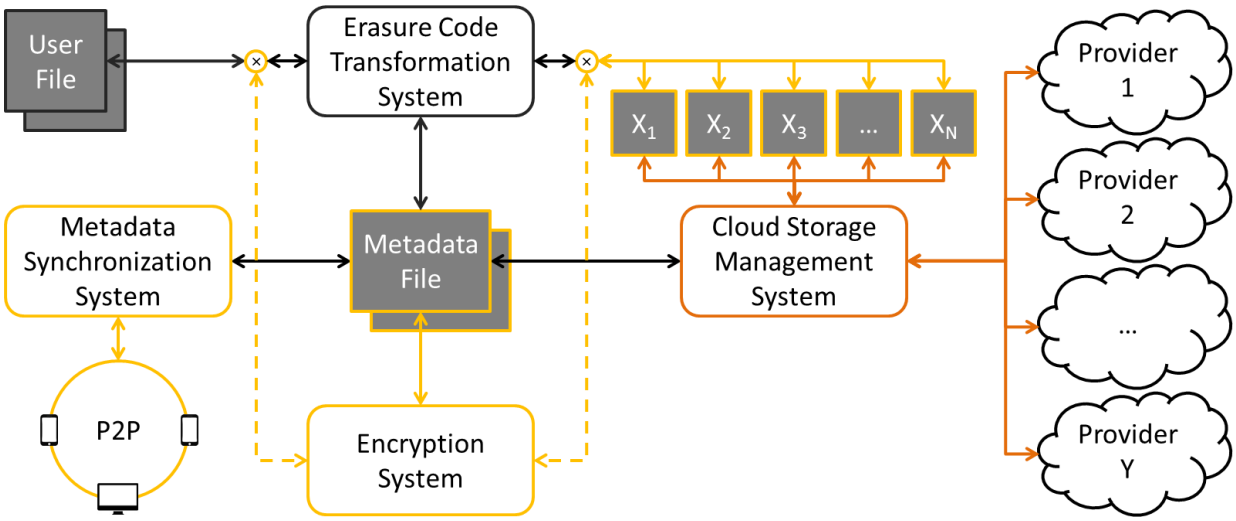

Figure 5 – General Model of Cloud Storage Framework

In fact, the choice of a storage provider could also include local and offline sources such as the user's hard drives, memory cards, rewritable optical disks, and USB drives. For the purposes of discussion we consider only Cloud providers as storage providers, but the Framework is not constrained to only use online providers. In fact the substitution can be advantageous in certain scenarios.

Since K out of N file pieces are required for the decoding transformation, the storage management function will always put fewer than K file pieces in a single Cloud storage provider for the security of the user's data. More strictly, the system will put fewer than R = (N – K) file pieces in each Cloud provider so that the complete loss of a provider does not affect the ability to recover or reconstruct the original file. The system can enforce a constraint that R < K, due to these two security and reliability principles.

The Framework will create some metadata in the form of a file which captures the information of which specific erasure code was used to transform the file, and which Clouds are storing the file pieces. Metadata files are synchronized across the user's devices by the Metadata Synchronization System through a peer-to-peer network consisting of only the user's devices. The metadata files are replicated across the peer-to-peer network so that the user always has the metadata on hand, along with the system software, to access their files on the Cloud. The system assumes that at any time, at least two peers are alive and can connect to each other to replicate the metadata.

Lastly, the Encryption System applies file level encryption to the metadata files, user files, and file pieces. Depending on the Framework configuration, encryption could either be applied before transformation operations. No matter what, the Framework requires file pieces to be encrypted prior to being uploaded to the Cloud.

6.2 Erasure Code Algorithm Properties and Metrics

The core component of the Framework is the erasure code used to encode and decode a file. This single component affects the security of the file, the potential costs to store the file, and the performance of the system. Since many erasure codes exist, it is worthy to analyze and compare a chosen representative sample of erasure codes to see their security properties, algorithmic properties, efficiencies, and theoretical performance limits in the domain of splitting and joining large size files. We focus on the algorithm aspect of each code in this section.

A set of erasure codes has been chosen based upon their popularity of use in industry, as well as their underlying mathematical principles. This selection criterion,

although not explicitly rigorous, follows the trends within the erasure codes research community. The majority of literature within this field can be traced back to a few canonical erasure codes, where earlier research works define and improve upon the codes from a theoretical point of view and latter research works improve the run-time performance of the codes and demonstrate their applications. The selection favors the canonical codes more than their refinements since the mathematics underlying the codes establish their algorithmic properties and theoretical performance limits. Comparing codes at these limits gives the Framework a rational means to pick an erasure code to use, given a user's situation or preference. Users may prefer to have their overall system run fast, which implies favor towards codes which encode and decode in linear time. Users may also prefer codes which are highly flexible, which implies favor towards codes that can be customized to produce any K number of redundant pieces relative to some division of the original file into M pieces. In practice, any implementation of the Framework can incorporate the latter refinements for each type of erasure code to maximize the system's performance.

The very first erasure code was invented by Richard Hamming in 1950 [48] using parity check in fixed positions to allow for either the detection or automatic correction of bit errors. Parity check is implemented in computers via the Exclusive OR (XOR, $\oplus$) operation. In fact, many modern erasure codes are based upon the use of XOR in different arrangements, for example RAID-5 erasure algorithm and Low Density Parity Code (LDPC). Shamir's Secret Sharing Algorithm is based on polynomial interpolation. Michael Rabin's Information Dispersal Algorithm (IDA) is based upon a matrix multiplication and matrix inversion process. Reed-Solomon codes are based upon polynomial multiplication and division over a Galois Field.

Recall from earlier in Section 2.2.1 a few important definitions. We defined K as the number of file pieces split from an original file, R as the number of redundancy file pieces added by an erasure code algorithm, and N as the total number of file pieces where N = K + R. Also, recall the definitions of Resultant Size Factor, Redundancy Factor, and the Redundancy Minimization Functions as follows:

$$Resultant\ Size\ Factor = \frac{K+R}{K} = \frac{N}{K}$$

$$Redundancy\ Factor = \frac{R}{N} = \frac{R}{K+R}$$

$$RSF \times RF = \frac{R}{K}$$

We can view the Framework as follows: for a given erasure code algorithm, a desired redundancy factor RF, and an appropriately chosen number of original file pieces K, the algorithm will add R redundancy file pieces.

With respect to a chosen RF, each algorithm can add a different number of redundant file pieces, or add them differently. The aim of Section 6.3 is to compare different algorithms for their algorithmic properties and theoretical performance at different chosen RF values, as an equalizing factor. RF directly corresponds to the security and reliability of the file, namely how many pieces can be lost in the set of N pieces before the file becomes unrecoverable through a decoding operation.

In this section, we begin by first defining a few constant properties which all algorithms share, then describe all comparison metrics point by point to show why these metrics are important to consider for the problem.

6.2.1 Common Algorithm Properties and Mathematical Constants

We use |F| to denote the size of the original file. A file F will be divided into K equal size pieces by any erasure code algorithm. We denote the size of each file piece as |K|. Logically, the size of each file piece is the size of the file divided by K. As such one constant holds throughout all erasure code algorithms:

Equation 5 – File Piece Size Constant

$$|K| = \frac{|F|}{K}$$

Further, while the number of redundancy file pieces can change for each algorithm, we hold the size of each piece constant and to be the same as each original file piece, that is:

Equation 6 – Redundancy File Size Constant

$$|R| = |K|$$

This constant property will become evident in Section 6.3 as we examine each algorithm in detail. We can further denote the total size of the resultant set of file pieces as |N|, which can be calculated as follows:

Equation 7 – Resultant File Size

$$|N| = |K| \times K + |R| \times R$$

6.2.2 Algorithm Analysis Metrics

There are 14 important metrics which we use to analyze each of the seven erasure codes in order to determine their properties, efficiencies, and suitability for solving the Cloud storage problem. The number of erasure codes in literature and in practice

can be countless. This set of metrics can be used by interested researchers and readers to analyze other erasure codes for their suitability to this problem.

Let us define the algorithm comparison metrics point by point as follows:

1. File Reconstruction Threshold

The file reconstruction threshold FT is the number of pieces within N needed to reconstruct the original file F. For most erasure code algorithms FT = K.

2. File Piece Reconstruction Threshold

The file piece reconstruction threshold PT is the minimum number of pieces within N needed to reconstruct one other piece within N.

3. Resultant Space

The resultant space is the total space taken up by a file after applying the encoding transformation to the file. It is the same as |N|, and is expressed in units of Bytes of computer data. Generally we will consider data of sizes in Megabytes (MB), Gigabytes (GB), Terabytes (TB), and Petabytes (PB) as these are the most prevalent size units found in 2014.

4. Resultant Space Factor

This is previously defined in Section 2.2.1. From a more intuitive standpoint, the resultant space factor is the size of the resultant file |N| divided by the size of the original file |F|. Since |N| is a function of |F|, we can express it as we have in Equation 1 in Section 2.2.1 as N ÷ K.

5. Redundancy Factor

This is also previously defined in Section 2.2.1. The redundancy factor is the number of redundancy file pieces R divided by the number of total number of file pieces N, expressed as a percentage.

6. Encoding Time

The encoding time is the amount of time necessary to encode a given file into its corresponding N file pieces. Encoding involves four steps of splitting the file into K pieces, and computing and creating R redundancy pieces. This is expressed as a function of |F|. Generally, a faster encoding time makes an algorithm better since it uses less CPU cycles, reducing resource consumption, and results in less waiting time for a user.

7. Decoding Time

The decoding time is the amount of time necessary to decode a given set of K file pieces back to the original file. This is also expressed as a function of |F|. A faster decoding time makes an algorithm better for the same reasons as Encoding Time.

8. Temporary Space for Encoding

Erasure code algorithms require a certain amount of memory or temporary file storage space to compute each encoding operation. This amount of space is defined as the Temporary Space for Encoding (TSE), expressed as a function of |K|. Lower TSE makes an algorithm more useful as it can be implemented in platforms which have less memory or disk space, such as a mobile phone.

9. Temporary Space for Decoding

Similarly, erasure code algorithms also require temporary memory or file spaces for the decoding operation. We define this space as the Temporary Space for Decoding (TSD), expressed as a function of |K|. Lower TSD is desirable for the same reasons as TSE.

10. Confusion Property

Recalling the definitions provided in Section 2.2.2, an erasure code algorithm has the confusion property if the output file pieces have complex mathematical relationships to the input file, and if none of the output file pieces directly correspond to the input pieces. For example, a code which computes some redundancy pieces and adds them to the original file would not be considered to have the Confusion Property because the output file contains the input file in plain text, without transforming it at all. Such types of codes do not have a complex relationship between the output and input files as far as the contents of the file are concerned.

11. Diffusion Property

An erasure code algorithm has the diffusion property if each bit of the input file is involved in computing all bits of each output file piece by the algorithm. Specifically, Diffusion is achieved if each input bit is mathematically involved in all output bits.

12. Partial Updates

An erasure code algorithm has this property if for a given update operation performed on the original file it does not need to recompute every output piece from scratch in order to update the output file pieces.

13. Metadata Requirement – Computation Key

A computation key is required by an erasure code algorithm if it must keep some numerical constants or data in the metadata file in order to mathematically perform the decoding procedure. An algorithm is more robust if it does not need a computation key, however this key can also offer an extra layer of security for the user since an attacker must also obtain the key before they can decode the file.

14. Metadata Requirement – File Reconstruction Relation Table

A file reconstruction relation table is required by an erasure code algorithm if it must use certain specific file pieces to reconstruct other file pieces in the set of N pieces, regardless of whether they are the original or redundancy pieces. More generally, an ordered reconstruction procedure is necessary for the algorithm to recompute the original file. An algorithm does not require this table if it does not require an ordered reconstruction. An algorithm is more robust if it does not require this table since it can begin reconstruction as soon as the first piece is downloaded, however it becomes easier for an attacker as well since it no longer needs to obtain specific file pieces to reconstruct the other specific pieces within the file set.

6.3 Algorithm Analysis

This section begins with an introduction of each erasure code algorithm along with their detailed analysis in Sections 6.3.1 through 6.3.7, and concludes with a comparative analysis of the algorithms in Section 6.3.8. For the purpose of a naive comparison, the “Simple Replication” algorithm is presented first to provide a basic reference point for all erasure code algorithms. Technically, Simple Replication is not an erasure code algorithm.

6.3.1 SIMPLE REPLICATION

To perform Simple Replication, users simply have to copy their original file as many times as they desire to achieve a particular redundancy factor. For example, if an RF of 50% is desired, the user copies their file once, such that 1 file out of the 2 is redundant. If an RF of 66% is desired, the user copies their file twice so that 2 files out of the 3 are redundant.

Its reconstruction thresholds PT and FT are always 1 since any copy is a true copy of the original. The resultant space is the number of copies time the size of the original file, thus $|N| = N \times |F|$; resulting in a RSF of $|N| \div |F| = N$. Its redundancy factor RF is $(N - 1) \div N$ as all pieces except one is redundant.

Mathematically speaking, every bit of an output file is the same as its corresponding bit in the input file. Thus, the encoding operation of creating N replicas involve a linear time operation to duplicate the data, and the encoding time is $O(|F|)$. The decoding operation requires no computation, so the decoding time is $O(1)$. The temporary memory space required for encoding is 0 since the operating system can manage the copy operation. The temporary memory space required for decoding is also 0.

Simple replication does not have confusion nor diffusion properties since an attacker can obtain the data if it obtains any copy of the file. Partial updates are supported since only the bits that are changed in an original file need to be updated in the duplicate copies. Simple replication does not need a computation key, and it does not need a file reconstruction relation table.

Table 10 shows the redundancy performance of using Simple Replication. As RF increases, the set of files contain more redundancy which means it is more tolerant to errors and losses. This is generally favorable for users who want more data security in their files. RSF shows the cost of attaining each level of data security, directly showing the amount of redundancy there is in the set of files. The ratio of RSF to RF shows the relative cost to attain each level of data security. Intuitively, using as few redundant copies as possible gives us the best performance, resulting in a lower RSF/RF ratio.

Table 10 – Simple Replication Configurations and Redundancy Performance

Simple Replication (N, K)	R	N	K	RSF	RF	RSF/RF
(2, 1)	**1**	**2**	**1**	**2.0000**	**0.5000**	**4.0000**
(3, 1)	2	3	1	3.0000	0.6667	4.5000
(4, 1)	3	4	1	4.0000	0.7500	5.3333
(5, 1)	4	5	1	5.0000	0.8000	6.2500
(6, 1)	5	6	1	6.0000	0.8333	7.2000
(7, 1)	6	7	1	7.0000	0.8571	8.1667
(8, 1)	7	8	1	8.0000	0.8750	9.1429
(9, 1)	8	9	1	9.0000	0.8889	10.1250
(10, 1)	9	10	1	10.0000	0.9000	11.1111

Simple Replication is trivially fast and allows partial updates. However, it costs a lot of storage space compared to the use of erasure code algorithms.

6.3.2 Hamming Code

Published in 1950, Richard Hamming introduced the world's very first erasure code while he was trying to solve the practical problem of allowing a system to automatically correct bit errors caused by analog data transmission, or noise [48]. His code is called the Hamming Code in literature, in honour of his name. In his original incarnation, known now as the (7, 4) Hamming Code, 3 error correction bits are

calculated and arranged for 4 data bits (D1, D2, D3, D4) according to the following table:

Table 11 – (7, 4) Hamming Code Computation Table

Bit Position	1	2	3	4	5	6	7
Bit Position Binary Value	1	10	11	100	101	110	111
Bit Type	Parity	Parity	Data	Parity	Data	Data	Data
Value	D1 ⊕ D2 ⊕ D4	D1 ⊕ D3 ⊕ D4	D1	D2 ⊕ D3 ⊕ D4	D2	D3	D4

It is known as the (7, 4) Hamming Code because in total there are 7 bits for every 4 data bits. Hamming Codes parity bits are set at any position that is a power of 2, that is bit position 2^m for $m = 0, 1, 2, \ldots$ Each parity bit correspondingly computes the XOR of all bit positions which have the m least significant bit set to 1.

In the example of Table 11, bit positions 3, 5, and 7 correspondingly have binary position values of 11, 101, and 111. The parity bit at position 1, with a corresponding m of 0, would calculate the XOR of all bits with the least significant bit having a value of 1, which is position 3, 5, 7. Hence, as shown, it computes the XOR of D1, D2, and D4. Similarly, the parity bit at position 2 with m = 1 computes the XOR of all bits with the second least significant bit having a value of 1, which is positions 3, 6, and 7.

The (7, 4) Hamming Code can correct a single error bit. For example if bit 5 (D2) was has an error (its bit value was flipped), then bits 4, 6, and 7 can be used to compute D2. Mathematically, (D2 ⊕ D3 ⊕ D4) ⊕ D3 ⊕ D4 = D2 since the XOR of any bit with itself is 0, and any bit XOR 0 is the value of that bit. The Code then uses the other parity bit which D2 is involved in to check that D2 was computed correctly, in this case parity bit 1. Let's denote the newly computed D2 value as D2'. It checks that P1 = D1 ⊕ D2 ⊕ D4 = D1 ⊕ D2' ⊕ D4. In this code, 3 bits are used to compute or

recompute another bit, and 4 bits are used to validate each bit through two equations.

Table 12 shows the computation table for the (15, 11) Hamming Code, where each parity bit is computed as the XOR of the data bits which are marked with an X.

Table 12 – (15, 11) Hamming Code Computation Table

Bit position		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Encoded data bits		p1	p2	d1	p4	d2	d3	d4	p8	d5	d6	d7	d8	d9	d10	d11
Parity bit coverage	p1	X		X		X		X		X		X		X		X
	p2		X	X			X	X			X	X			X	X
	p4				X	X	X	X					X	X	X	X
	p8								X	X	X	X	X	X	X	X

Generally, a Hamming Code with m parity bits will allow for $2^m - m - 1$ data bits, and the total number of bits is $2^m - 1$. Valid Hamming Code arrangements include: (3, 1), (7, 4), (15, 11), (31, 26), (63, 57). Although the (3, 1) Hamming Code is technically a triple repetition code which has the characteristics of Simple Replication. For every extra parity bit, the total number of bits roughly doubles, and the number of bits involved in every parity bit calculation also roughly doubles. In fact, every parity bit involves $2^{(m-1)} - 1$ data bits in its calculation. For any two parity bits, their equations will have $2^{(m-2)}$ bits overlapping, which is visually evident in Table 12.

To use Hamming Code in the Framework, a file F can be split into K data pieces and then Hamming Code would be computed bitwise by taking a bit from each data piece one at a time. For example, to use the (7, 4) Hamming Code, we would split the file into 4 pieces. In each iteration, a bit from each of the four pieces, K1, K2, K3, and K4, is taken to compute 3 parity bits R1, R2, and R3 which are written into the 3 redundancy file pieces in order.

The encoding time is linear with respect to the file size, or simply O(|F|), since XOR is implemented in the CPU in hardware and the parity bits can be computed through a single pass read through all the data bits. To check for errors, every parity bit needs to be recomputed from their data bits and checked against the stored parity bit. If no errors are present in the set of file pieces, then decoding involves the proper sequential arrangement of all data bits. Hence, decoding operation for a set of file pieces that have no errors would be O(|F|). If errors exist, then two equations are necessary to recover and validate a bit, assuming all bits involved except the error bit is correct. The error bit must also be present in both equations, hence $2^{(m-1)} - 1 + 2^{(m-2)} = 3 \times 2^{(m-2)} - 1$ bits are required to recover an error bit. In such scenarios, the computation necessary to recover 1 bit involves computing the bit through a single pass of reading through the bits of one equation, and then validating the bit through another equation. Decoding time is thus O(|F|) overall.

Hamming Code has a file reconstruction threshold FT = K, a file piece reconstruction threshold PT = $2^{(m-1)} - 1$, a resultant space of $2^m - 1$, a resultant space factor of $(2^m - 1) \div (2^m - m - 1)$, and a redundancy factor of $m \div (2^m - m - 1)$.

Table 13 – Hamming Codes and Redundancy Performances

Hamming Code (N, K)	Parity Bits (R)	Total Bits (N)	Data Bits (K)	Bits needed to recover 1 other bit	Bit overlap from any 2 equations	Bits needed to validate another bit	RSF	RF	RSF/RF
(3, 1)	2	3	1	1	2	1	3.0000	0.6667	4.5000
(7, 4)	3	7	4	3	5	2	1.7500	0.4286	4.0833
(15, 11)	4	15	11	7	11	4	1.3636	0.2667	5.1136
(31, 26)	5	31	26	15	23	8	1.1923	0.1613	7.3923
(63, 57)	6	63	57	31	47	16	1.1053	0.0952	11.6053
(127, 120)	7	127	120	63	95	32	1.0583	0.0551	19.2012
(255, 247)	8	255	247	127	191	64	1.0324	0.0314	32.9074
(511, 502)	9	511	502	255	383	128	1.0179	0.0176	57.7957
(1023, 1013)	10	1023	1013	511	767	256	1.0099	0.0098	103.3099

Table 13 shows the redundancy performance of some of the Hamming Codes. As we increase the number of parity bits used in the code, both RSF and RF approach 1 and 0 respectively.

It is not surprizing why the (7, 4) Hamming Code was the most popular among the set, as well as being the canonical code, as it achieves the best trade off of redundancy and resultant space without resorting to the simple replication of the (3, 1) Hamming Code. Higher level Hamming Codes reduce the safety of the set of data as there are relatively fewer number of parity bits compared to data bits. The table confirms this as RF drops to nearly 0 for higher level Hamming Codes.

The temporary memory space needed for encoding and decoding is m + k bits. Relative to files in MB or higher in size, we can safely claim that the memory space needed is negligible. If some data bit changes upon an edit, then its corresponding parity bits must be updated. Hamming Code supports partial updates since not every parity bit must change for a given data bit change.

Hamming Code does not have either the confusion or diffusion properties, since attackers who gain sufficient numbers of bits necessary to construct another bit can then progressively work towards gaining the data of the entire system by reconstructing one bit at a time. As the data bits are written to the output pieces unchanged, an attacker can at times guess any missing bits using dictionary attacks.

Lastly, Hamming Code does require the use of a reconstruction relation table in order to keep track of the file pieces and their corresponding bit position in the code table. It does not need a computation key however.

6.3.3 RAID-5 ALGORITHM

Redundant Array of Independent Disks (RAID) is a suite of stream or block based redundancy algorithms in addition to hard drive configurations, invented by Peter M. Chen, et al. in 1994 [49]. RAID is implemented in almost all modern computer systems as a disk-level data redundancy system, working behind the scenes to protect a users' data. The series of RAID algorithms starting with RAID-1 was invented to provide redundancy for storage of data in disks. We are interested in the algorithms only, and in particular the RAID-5 single parity algorithm.

RAID-5 stripes all data at the block level into K number of pieces (A, B, C, D, …, K), while maintaining identical sizes for each piece. It then computes one parity piece (R) with the size equal to one of the pieces, each bit within the parity piece is the result of the XORs of the corresponding bit from the split pieces. The equation to compute P is as follows:

Equation 8 – RAID-5 Encoding Algorithm

$$\mathbf{R_i} = A_i \oplus B_i \oplus C_i \oplus \ldots \oplus K_i, \text{ for } i = 0 \text{ to } |K|, \text{ given } |K| \text{ in bits}$$

The parity piece affords at most one piece to fail among any of the pieces. The recovery of a missing piece is done with the same procedure; however, we compute the missing piece bit by bit by taking the XOR of all remaining pieces. For example if piece B was completely erased, we could recompute it as follows:

Equation 9 – RAID-5 Bit Repair Algorithm

$$\mathbf{B_i} = \mathbf{R_i} \oplus A_i \oplus C_i \oplus \ldots \oplus K_i, \; i = 0 \text{ to } |K|, \text{ given } |K| \text{ in bits.}$$

We can show that given a file of size |F| and the desired the number of piece K, RAID-5 will produce a total output file size |N| = |F| + |F| ÷ K. This has a resultant space factor of (K + 1) ÷ K, and a redundancy factor of 1 ÷ (K + 1). RAID-5 requires that all pieces except 1 be present in order to reconstruct the missing or erroneous piece. Its file reconstruction threshold and file piece reconstruction threshold are both K.

Implementing RAID-5 algorithm in software would require the memory space of (K + 1) × |K| bits. For encoding a file, the program will read K bits at a time and split them into the K memory blocks. Then the program computes the Parity block from these blocks, and writes all K + 1 blocks to the corresponding output files. The program continues until all bits from the file have been read. RAID-5 encodes a file in time O(|F|), using memory space of O(K + 1) bits. Reconstruction of the original file in RAID-5 simply requires reassembling the original file by reading the bits from all K file pieces in the correct order. RAID-5 decodes a file in time O(|F|) using memory space of O(K) bits. Like Hamming Code, RAID-5 uses relatively negligible memory space.

Table 14 – RAID-5 Schemes and Redundancy Performances

RAID-5 (N, K)	R	N	K	RSF	RF	RSF/RF
(2, 1)	1	2	1	2.0000	0.5000	4.0000
(3, 2)	**1**	**3**	**2**	**1.5000**	**0.3333**	**4.5000**
(4, 3)	1	4	3	1.3333	0.2500	5.3333
(5, 4)	1	5	4	1.2500	0.2000	6.2500
(6, 5)	1	6	5	1.2000	0.1667	7.2000
(7, 6)	1	7	6	1.1667	0.1429	8.1667
(8, 7)	1	8	7	1.1429	0.1250	9.1429
(9, 8)	1	9	8	1.1250	0.1111	10.1250
(10, 9)	1	10	9	1.1111	0.1000	11.1111

RAID-5 can correct errors in at most 1 block, through reconstruction of the block. A checksum must be computed for each block in order to use it as an indicator of

whether the block has been modified. RAID-5 is similar to Hamming Code in that one single parity equation is used throughout the system.

Table 14 shows a sample of RAID-5 schemes and their performances. The (3, 2) RAID-5 scheme is the most popular as it gives the best ratio between RSF and RF without resorting to simple replication in the (2, 1) scheme.

From a security point of view, RAID-5 does not achieve the properties of diffusion and confusion. Like Hamming Code, attackers can use the dictionary attack to break this code. RAID-5 supports partial updates as changes in any data bit require only a corresponding update to that parity bit. It does not require a computation key but does require a reconstruction table to identify the ordering of the original file pieces.

6.3.4 Low-Density Parity-Check Codes

Low-Density Parity-Check Codes (LDPC) were invented by Robert G. Gallagher in 1960 [50] as part of his doctoral dissertation, subsequently published in 1963. LDPC is similar to RAID-5 in that both use the XOR operation. In LDPC, given a file F split into K pieces, R additional redundancy pieces are computed, each by taking a subset of the K pieces and computing their XOR parity. A specific configuration of LDPC and example is as follows:

Let the K pieces be K1, K2, K3 and K4, and the R redundancy pieces be R1, R2, R3, R4. Each of the blocks have size |F| ÷ K. Then:

$$R1 = K1 \oplus K2 \oplus K3$$

$$R2 = K1 \oplus K2 \oplus K4$$

$$R3 = K1 \oplus K3 \oplus K4$$

$$R4 = K2 \oplus K3 \oplus K4$$

Once computed, the equations can be rewritten in the form:

$$0 = K1 \oplus K2 \oplus K3 \oplus R1$$

$$0 = K1 \oplus K2 \oplus K4 \oplus R2$$

$$0 = K1 \oplus K3 \oplus K4 \oplus R3$$

$$0 = K2 \oplus K3 \oplus K4 \oplus R4$$

In this set of equations, any 3 of 8 total pieces can be used to reconstruct 1 other piece just like the process in RAID-5. We also only need half the number of total pieces to reconstruct all pieces, such as by obtaining N1, N2, R1, and R2 which can then be used to compute N3 and N4. If no pieces were lost, we can optimally choose the pieces which cost less to download and reconstruct the pieces which are more expensive to download. These features make it better than RAID-5.

A reconstruction relation table must be kept for LDPC in order to keep track of the relationship between each data piece and its corresponding redundancy pieces. However, assuming all pieces have no errors, the system could try all combinations of pieces until it finds the subsets which yield a chained XOR result of 0. One piece out of each chain must be a redundancy piece, as shown in the above equation sets, and could be logically deduced from examining the involvement of the other pieces in other equations. For an attacker, breaking the system this way is much more expensive than trying to obtain the metadata file containing the relation table. When K is chosen to be very large, trying all combinations becomes very costly.

For any LDPC configuration, the total time to encode the file into K data pieces and to compute the R redundancy pieces is O(|F|). The key value to choose in LDPC is the number of data pieces PT involved in the calculation of each redundancy piece R, where PT ≤ K. One might observe that LDPC follows the binomial theorem to choose and permute the data pieces used to compute each of the redundancy pieces. Thus, R can be computed as a function of PT and K as follows:

Equation 10 – Binomial Coefficient Formula for LDPC Codes

$$R = \binom{K}{PT}$$

The threshold number of pieces to reconstruct the original file is FT = K. It results in a space of |F| + |R| × R, with a redundancy factor of R ÷ N. The Resultant Space Factor is N ÷ K. Similar to RAID-5, we can formulate R by computing using PT bits of each of the K pieces at a time and then writing the output to the R files sequentially, so the temporary space required to encode is R × K × PT bits. It is relatively more than Hamming Code and RAID-5, but still negligible considering files in sizes of MB or higher. The total time to decode a file is O(|F|) with the fastest by simply reading and joining the K data pieces, and the slowest by reconstructing some of the K data pieces using the R redundancy pieces. The temporary space required for decoding is |F|.

We know that (K choose PT) ≥ K, however the total number of redundancy pieces R does not necessarily have to match the total number of possible permutations. For example, let the K pieces be K1, K2, K3, and K4, and the redundancy pieces be R1, R2 computed as follows:

$$R1 = K1 \oplus K2 \oplus K3$$

$$R2 = K2 \oplus K3 \oplus K4$$

To recover any piece, only three pieces are needed. However the relative risk and cost of losing each piece is not the same for systematic total reconstruction. For example, if K2 and K3 were lost at the same time then the system cannot possibly reconstruct either of them, however if K1 and K4 were lost at the same time the system could reconstruct both of them. In short, rather than only requiring any half of the number of pieces to reconstruct the file, this now requires some chosen subset. This lets the system choose different redundancy factors and have a different resultant space, while maintaining the same time complexity and roughly the same temporary space requirements.

We will utilize all possible R combinations for performance analysis, as maintaining the property of letting any K number of pieces be used for reconstruction gives substantially higher flexibility in data placement in the Cloud. For these, the resultant space is ((K choose PT) + K) × |F| ÷ K. The resultant size factor is ((K choose PT) + K) ÷ K. The redundancy factor is ((K choose PT) + K) ÷ (K choose PT).

Table 15 shows some possible configurations for LDPC, where PT is set with respect to K shown in the leftmost column. When PT = K, the system behaves the same as RAID-5. When PT = K – 1, the system constantly produces configurations which result in RSF of 2 and RF of 0.5; that is there is an equal number of redundancy file pieces as there are original file pieces. When K ÷ 2 ≤ PT ≤ K – 2, LDPC generates more redundancy pieces than the number of file pieces in the system but in a more controlled fashion than simple replication.

Table 15 – LDPC Configurations and Redundancy Performances

	LDPC (N, K, PT)	PT	R	N	K	RSF	RF	RSF/RF
PT = K	(2, 1, 1)	1	1	2	1	2.0000	0.5000	4.0000
	(3, 2, 2)	2	1	3	2	1.5000	0.3333	4.5000
	(4, 3, 3)	3	1	4	3	1.3333	0.2500	5.3333
	(5, 4, 4)	4	1	5	4	1.2500	0.2000	6.2500
	(6, 5, 5)	**5**	**1**	**6**	**5**	**1.2000**	**0.1667**	**7.2000**
	(7, 6, 6)	6	1	7	6	1.1667	0.1429	8.1667
	(8, 7, 7)	7	1	8	7	1.1429	0.1250	9.1429
	(9, 8, 8)	8	1	9	8	1.1250	0.1111	10.1250
	(10, 9, 9)	9	1	10	9	1.1111	0.1000	11.1111
PT = K - 1	(4, 2, 1)	1	2	4	2	2.0000	0.5000	4.0000
	(6, 3, 2)	2	3	6	3	2.0000	0.5000	4.0000
	(8, 4, 3)	3	4	8	4	2.0000	0.5000	4.0000
	(10, 5, 4)	4	5	10	5	2.0000	0.5000	4.0000
	(12, 6, 5)	5	6	12	6	2.0000	0.5000	4.0000
	(14, 7, 6)	6	7	14	7	2.0000	0.5000	4.0000
	(16, 8, 7)	7	8	16	8	2.0000	0.5000	4.0000
	(18, 9, 8)	8	9	18	9	2.0000	0.5000	4.0000
	(20, 10, 9)	9	10	20	10	2.0000	0.5000	4.0000
PT = K - 2	(6, 3, 1)	1	3	6	3	2.0000	0.5000	4.0000
	(10, 4, 2)	2	6	10	4	2.5000	0.6000	4.1667
	(15, 5, 3)	3	10	15	5	3.0000	0.6667	4.5000
	(21, 6, 4)	4	15	21	6	3.5000	0.7143	4.9000
	(28, 7, 5)	5	21	28	7	4.0000	0.7500	5.3333
	(36, 8, 6)	6	28	36	8	4.5000	0.7778	5.7857
	(45, 9, 7)	7	36	45	9	5.0000	0.8000	6.2500
	(55, 10, 8)	8	45	55	10	5.5000	0.8182	6.7222
	(66, 11, 9)	**9**	**55**	**66**	**11**	**6.0000**	**0.8333**	**7.2000**
PT = K - 3	(8, 4, 1)	1	4	8	4	2.0000	0.5000	4.0000
	(15, 5, 2)	2	10	15	5	3.0000	0.6667	4.5000
	(26, 6, 3)	3	20	26	6	4.3333	0.7692	5.6333
	(42, 7, 4)	**4**	**35**	**42**	**7**	**6.0000**	**0.8333**	**7.2000**
	(64, 8, 5)	5	56	64	8	8.0000	0.8750	9.1429
	(93, 9, 6)	6	84	93	9	10.3333	0.9032	11.4405
	(130, 10, 7)	7	120	130	10	13.0000	0.9231	14.0833
	(176, 11, 8)	8	165	176	11	16.0000	0.9375	17.0667
	(232, 12, 9)	9	220	232	12	19.3333	0.9483	20.3879

Examining the three configurations with an equivalent performance through the RSF/RF ratio of 7.2, we can see the flexibility of LDPC. To achieve this, LDPC could split the file into 5 pieces and compute 1 redundancy piece, shown in the (6, 5, 5) configuration. It could also split the file into 11 pieces and compute 55 redundancy pieces using 9 pieces at a time, as shown in the (66, 11, 9) configuration. Lastly, it

could split the file into 7 pieces and compute 35 redundancy pieces using 4 pieces at a time, as shown in the (42, 7, 4) configuration.

Like RAID-5, the system must download all but 1 piece to recover the file in the (6, 5, 5) configuration, however the other two configurations offer substantially higher flexibility. In (66, 11, 9) configuration, the least amount of pieces it needs to download is 11 out of 66 total pieces. In (42, 7, 4) configuration, the least amount of pieces required is 7 out of 42. In the latter two configurations, the absolute cost is higher as the total file size is 6 times the original, but it grants this flexibility during any recovery operation. Shown through their RF values of 0.8333, they both are more secure than the (6, 5, 5) configuration which has an RF of 0.1667.

From a security point of view, LDPC achieves diffusion but not confusion as given a sufficient subset of K pieces, the remaining pieces could be guessed using the dictionary attack. Partial updates are supported by LDPC as only the bits in the file pieces corresponding to the modified bits in the original file need to be recomputed.

6.3.5 Shamir's Secret Sharing Algorithm

Adi Shamir introduced a secret sharing scheme and algorithm in 1979 in his publication "How to share a secret" in Communications of the ACM [51]. The scheme is a threshold scheme, and is most widely used to split a master encryption key into shares whereby any subset of shares meeting the minimum threshold can be used to reconstruct the master key.

The principle behind the algorithm is that for any polynomial of degree K – 1 requires K points to define. We choose one particular point of a degree K – 1 equation to be the secret point where it's Y value is the binary numeric value of the data, and

we randomly choose N other points (N > K) from the polynomial as shares to be stored. The secret can only be computed when the polynomial can be reconstructed, which requires obtaining at least K out of N points to be used in curve fitting for reconstructing the equation. The Y values of the N points share the same range as the Y value of the secret, and in binary means they have the same number of bits. As such, each share has the same file size as the original file.

Given a file size of |F|, Shamir's Secret Sharing algorithm will produce a total output file size of $N \times (|F| + C)$ where C is a very small constant amount of data related to each share. This has a redundancy factor of $(N - K) \div N$. Relative to the size of the file however, the resultant space factor is $N \div 1 = N$ since every point has the same size as the secret point.

Given an input file F, it is divided in to a sequence of fixed sized *m*-bit pieces. Starting with the first piece, the binary value of the piece is taken as integer number y and assigned as the secret point of that piece, typically (X = 0, Y = y). y has a range of $[0, 2^m)$. N points are chosen by picking at random their X values, as long as it does not equal the X value of the secret point. These X values are used for all pieces. Their Y values are calculated per piece. Each resultant piece would contain one single X coordinate, and a list of Y coordinates each corresponding to a K – 1 degree equation. Figure 6 shows an example of an equation of degree 3, with 13 gray points defined on the curve. The black point at X = 0 is the secret.

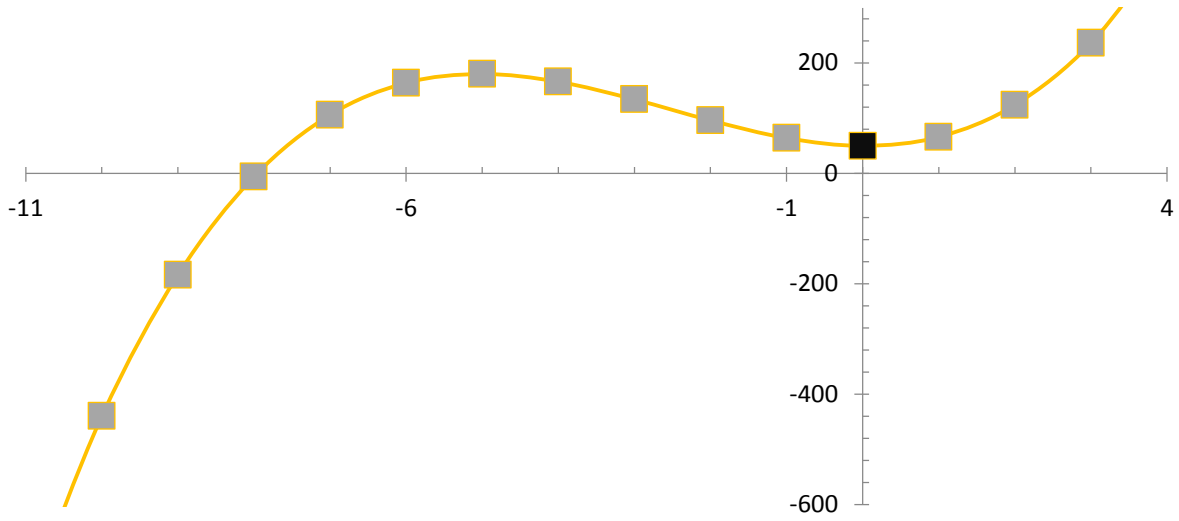

Figure 6 – A Polynomial Equation of Degree 3 with 13 Points Defined

For each equation, K – 1 random integers are chosen to form a K – 1 degree equation, whereby the constant term in the equation is calculated by putting the secret point into the equation and solving for Y. For example, let's create a degree 2 polynomial:

$$Y = C_1X^2 + C_2X + C$$

Let the secret be the number 25, and we assign it to the point (X = 3, Y = 25). By random we select two integers 3 and 9 for the constants, so the equation looks like the following after we substitute in the two integers:

$$Y = 3X^2 + 9X + C$$

To solve for C, we put the secret point into the equation:

$$25 = 3 \times 3^2 + 9 \times 3 + C$$

$$25 = 27 + 27 + C$$

$$C = 25 - 54 = -29$$

The finalized equation is:

$$Y = 3X^2 + 9X - 29$$

The N points of the block are obtained by randomly choosing N distinct X coordinates and calculating their corresponding Y values using the equation. The N points are written to their corresponding output files. Continuing the example, let us suppose we chose 4 coordinate points of -4, -3, 0, and 2:

$$Y_1 = 3 \times (-4)^2 + 9 \times (-4) - 25 = -17$$

$$Y_2 = 3 \times (-3)^2 + 9 \times (-3) - 25 = -29$$

$$Y_3 = 3 \times 0^2 + 9 \times 0 - 25 = -29$$

$$Y_4 = 3 \times 2^2 + 9 \times 2 - 25 = 1$$

Thus the redundancy data points are (–4, –17), (–3, –29), (0, –29), and (2, 1). The equation, secret point, and redundancy points are plotted in Figure 7.

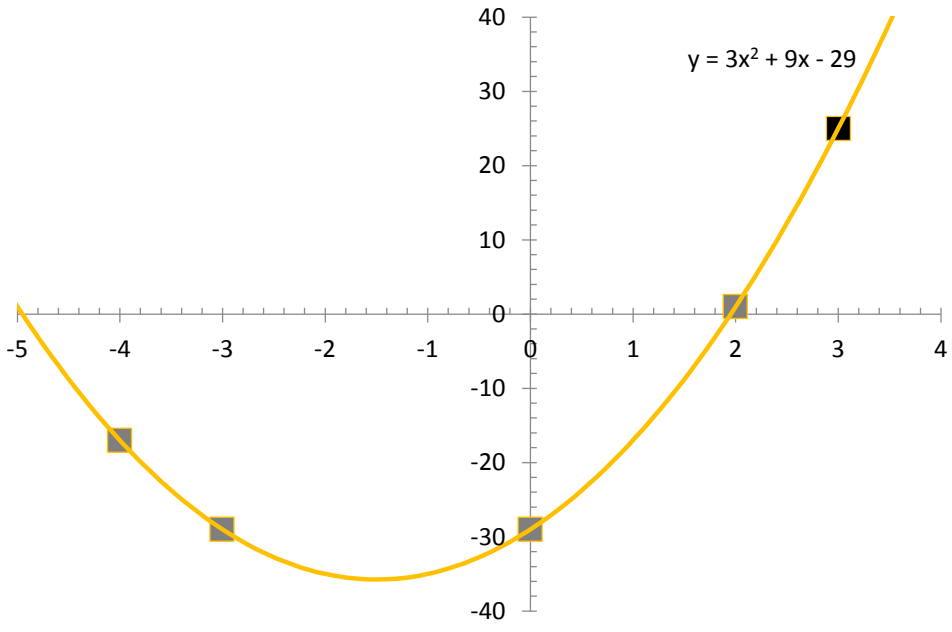

Figure 7 – Shamir's Secret Sharing Example

Once all pieces are processed, the X values of the N points and the secret point, as well as the piece size m, are written to the metadata file. This file forms the Key (analogous to encryption Key) for recovering the file. Shamir's Secret Sharing algorithm encodes a file in time O(|F|), although slower than RAID-5, and uses memory space of O(N × m) bits.

Given any K shares, the reconstruction process begins by reading into program memory the X value and piece size m from the Key file. The equation is recovered by computing for the secret equation F(x) piece by piece from the K share files. For each piece read into memory, F(x) is obtained by first computing the set of LaGrange basis polynomials $L_j(x)$:

Equation 11 – LaGrange Basis Polynomials Equation

$$L_j(x) = \prod_{\substack{0 \le m \le T \\ m \ne j}} \frac{x - x_m}{x_j - x_m} = \frac{(x - x_m)}{(x_j - x_m)} \cdots \frac{(x - x_{j-1})}{(x_j - x_{j-1})} \frac{(x - x_{j+1})}{(x_j - x_{j+1})} \cdots \frac{(x - x_T)}{(x_j - x_T)}$$

Then the equation is computed as:

$$F(x) = \sum_{j=0}^{T} y_j . L_j(x)$$

The secret point is then calculated from the equation. Each basis polynomial has K numerator and denominator terms. The equation is the sum of the K basis polynomials times the y value of the N points. Thus, Shamir's Secret Sharing algorithm reconstructs a file in time $O((K \times |F|)^2)$ and uses memory space of $O((K \times m)^2)$ bits.

If up to R = N – K shares are corrupted or destroyed, they can still be recomputed by utilizing the K shares to formulate the equation, then picking at random R other X values. Shamir's Secret Sharing algorithm requires K valid shares to repair any corrupted shares, and like RAID-5 will need to compute checksums for each share in order to ascertain whether the share has been modified.

Table 16 – Shamir's Secret Sharing Schemes and Redundancy Performance

Shamir's (N, K, PD)	Polynomial Degree (PD)	Points to define Equation (K)	Redundant Equation Points (R)	Total Points (N)	RSF	RF	RSF/RF
(2, 2, 1)	1	2	0	2	2.0000	0.0000	Undefined
(3, 2, 1)			1	3	3.0000	0.3333	9.0000
(4, 2, 1)			**2**	**4**	**4.0000**	**0.5000**	**8.0000**
(5, 2, 1)			3	5	5.0000	0.6000	8.3333
(6, 2, 1)			4	6	6.0000	0.6667	9.0000
(7, 2, 1)			5	7	7.0000	0.7143	9.8000
(8, 2, 1)			6	8	8.0000	0.7500	10.6667
(3, 3, 2)	2	3	0	3	3.0000	0.0000	Undefined
(4, 3, 2)			1	4	4.0000	0.2500	16.0000
(5, 3, 2)			2	5	5.0000	0.4000	12.5000
(6, 3, 2)			**3**	**6**	**6.0000**	**0.5000**	**12.0000**
(7, 3, 2)			4	7	7.0000	0.5714	12.2500
(8, 3, 2)			5	8	8.0000	0.6250	12.8000
(9, 3, 2)			6	9	9.0000	0.6667	13.5000
(4, 4, 3)	3	4	0	4	4.0000	0.0000	Undefined
(5, 4, 3)			1	5	5.0000	0.2000	25.0000
(6, 4, 3)			2	6	6.0000	0.3333	18.0000
(7, 4, 3)			3	7	7.0000	0.4286	16.3333
(8, 4, 3)			**4**	**8**	**8.0000**	**0.5000**	**16.0000**
(9, 4, 3)			5	9	9.0000	0.5556	16.2000
(10, 4, 3)			6	10	10.0000	0.6000	16.6667

To use Shamir's Secret Sharing, a polynomial degree must be chosen for the equation, as well as the desired number of redundant points on the equation. Table 16 shows various configurations of Shamir's Secret Sharing. Shamir's Secret Sharing seems to be most efficient when the total number of points N is double of the number of points K needed to reconstruct each equation of degree K – 1. Examples include (4, 2, 1), (6, 3, 2), and (8, 4, 3) configurations. Overall, using lower polynomial degree

equations result in a more efficient system. We can see that with an RSF of 8, using a polynomial degree 1 equation gives us a redundancy factor of 0.75 while it gives a redundancy factor of 0.625 and 0.5 using polynomial degree 2 and 3 equations, respectively.

From a security point of view, Shamir's Secret Scheme algorithm achieves both the properties of Confusion and Diffusion. The original text is transformed into a set of N file pieces which is tied by a mathematical relationship that can only be reverse engineered if the Key file was decrypted, and if K out of N file pieces were obtained. An attacker which obtains a number of files less than K cannot guess the contents in the remaining files since they cannot reconstruct the equation.

Overall Shamir's Secret Scheme has a higher space cost than the previous studies erasure codes. Its mathematical technique must represent and bound each secret point's data value to a range in order to control the resulting binary file size. Since every point on the equation shares this bounded range, the more points needed the higher the cost in terms of resultant file space. It is however much more secure, since it is immune to dictionary attacks.

6.3.6 Rabin's Information Dispersal Algorithm

Michael Rabin's Information Dispersal Algorithm (IDA) was brought to the world in his paper published in the Journal of ACM in 1989 [52]. It is also an erasure code, and is based on a matrix multiplication and inversion process.

IDA considers a file to have L symbols. IDA first splits the L symbols into $M = \lceil L \div K \rceil$ input fragments, where K is the number of symbols per fragment. Each fragment has size $|F| \div M$ in bits. If L is not a multiple of K, then IDA pads the message with zeroes

to generate extra symbols. Let us denote these input fragments as F_i. Thus, the file will be split into fragments $\{F_0, F_1, F_2, \ldots, F_{[L \div M]-1}\}$ [58].

IDA uses an $N \times K$ encoding matrix, denoted as A, where any K rows of the matrix are linearly independent. One type of matrix which satisfies this requirement is a Vandermonde matrix, where for row i the values of the matrix are:

$$1, (i + 1), (i + 1)^2, (i + 1)^3 \ldots (i + 1)^{K-1}$$

To encode, IDA takes each input fragment F_i one at a time and multiplies it to matrix A to form the output fragments. Then, the fragments are organized column wise into an output matrix. Each row of this matrix forms an output file piece. IDA outputs N file pieces, each piece having M symbols. Any K of N file pieces can be used to recover the original message.

For example, let there be a message have 10 integers (1, 3, 5, 2, 4, 6, 7, 8, 9, 11) and we want to split it into 4 fragments. In this case, L = 10 and K = 3. Since L is not an integer multiple of K, we pad the message with zeroes. $M = [L \div K] = 4$ for this example. In order they are (1, 3, 5), (2, 4, 6), (7, 8, 9), and (11, 0, 0) respectively.

We construct the encoding matrix as a Vandermonde matrix A, with N = 8:

$$A = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 4 \\ 1 & 3 & 9 \\ 1 & 4 & 16 \\ 1 & 5 & 25 \\ 1 & 6 & 36 \\ 1 & 7 & 49 \\ 1 & 8 & 64 \end{bmatrix}$$

The output fragments are computed by taking each input fragment and multiply it with matrix A:

$$[1 \quad 3 \quad 5] \times A = \begin{bmatrix} 9 \\ 27 \\ 55 \\ 93 \\ 141 \\ 199 \\ 267 \\ 345 \end{bmatrix} \quad [2 \quad 4 \quad 6] \times A = \begin{bmatrix} 12 \\ 34 \\ 68 \\ 114 \\ 172 \\ 242 \\ 324 \\ 418 \end{bmatrix}$$

$$[7 \quad 8 \quad 9] \times A = \begin{bmatrix} 24 \\ 59 \\ 112 \\ 183 \\ 272 \\ 379 \\ 504 \\ 647 \end{bmatrix} \quad [11 \quad 0 \quad 0] \times A = \begin{bmatrix} 11 \\ 11 \\ 11 \\ 11 \\ 11 \\ 11 \\ 11 \\ 11 \end{bmatrix}$$

Treating each fragment as a column, we join the fragments together to form the N × M output matrix:

$$\begin{bmatrix} 9 & 12 & 24 & 11 \\ 27 & 34 & 59 & 11 \\ 55 & 68 & 112 & 11 \\ 93 & 114 & 183 & 11 \\ 141 & 172 & 272 & 11 \\ 199 & 242 & 379 & 11 \\ 267 & 324 & 504 & 11 \\ 345 & 418 & 647 & 11 \end{bmatrix}$$

Each output file piece is a row of the output matrix, for this example it is (9, 12, 24, 11), (27, 34, 59, 11) … (345, 418, 647, 11). To reconstruct the original message, any K file pieces will suffice, for example taking the 3 pieces (9, 12, 24, 11), (55, 68, 112, 11), and (93, 114, 183, 11) which corresponds to rows 1, 3, and 4 of the output matrix.

We construct the K × K decoding matrix by copying the rows from the encoding matrix A that correspond to each file piece:

$$B = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 3 & 9 \\ 1 & 4 & 16 \end{bmatrix}$$

We compute the inverse matrix of B:

$$B^{-1} = \frac{1}{6}\begin{bmatrix} 12 & -12 & 6 \\ -7 & 15 & -8 \\ 1 & -3 & 2 \end{bmatrix}$$

Also, we arrange the output pieces row wise corresponding to the decoding matrix:

$$\begin{bmatrix} 9 & 12 & 24 & 11 \\ 55 & 68 & 112 & 11 \\ 93 & 114 & 183 & 11 \end{bmatrix}$$

Taking each column in order from left to right and multiplying it to the inverse matrix of B, we get each original message fragment:

$$\begin{bmatrix} 9 \\ 55 \\ 93 \end{bmatrix} \times B^{-1} = [1 \quad 3 \quad 5] \quad \begin{bmatrix} 12 \\ 68 \\ 114 \end{bmatrix} \times B^{-1} = [2 \quad 4 \quad 6]$$

$$\begin{bmatrix} 24 \\ 112 \\ 183 \end{bmatrix} \times B^{-1} = [7 \quad 8 \quad 9] \quad \begin{bmatrix} 11 \\ 11 \\ 11 \end{bmatrix} \times B^{-1} = [11 \quad 0 \quad 0]$$

The time it takes to encode a file is $O(|F|^2)$ due to the matrix multiplication operation, while the time to decode a file is $O(|F|^3 + |F|^2)$ due to the matrix inversion operation on B, and the subsequent multiplication operation. The matrix A is the Key for IDA and must be stored in the metadata file. The matrix can have a size of a fraction of |F| since they are just numeric coefficients. The temporary memory space required for encoding the file is around O(|F|) since we must store the matrix in the memory but

can read the file and write out its encoding to the output file one piece at a time. Each piece has M symbols and each symbol is |F| ÷ L bits long, so the temporary memory space needed for encoding the file is O(|F| + |F| × M ÷ L) bits. The temporary memory space required to decode the file is also O(|F| + |F| × M ÷ L), where O(|F|) space is used to store the inverse matrix, and |F| × M ÷ L bits is used as input and output file buffers. The set of output files has a total space of N × M × |F| ÷ L bits. However an additional O(|F|) space must be used to store the metadata. The resultant size factor is N × M ÷ L. The redundancy factor is (N – K) ÷ N.

Table 17 – Rabin's IDA Configurations and Redundancy Performance

	Rabin's IDA (N, K, L, M)	Symbols / File (L)	Fragments (M)	Extra Symbols to Fill Fragments	R	N	Symbols / Fragment (K)	RSF	RF	RSF/RF
Increasing K	(10, 2, 10, 5)	10	5	0	8	10	2	5.0000	0.8000	6.2500
	(10, 3, 10, 4)	10	4	2	7	10	3	4.0000	0.7000	5.7143
	(10, 4, 10, 3)	10	3	2	6	10	4	3.0000	0.6000	5.0000
	(10, 5, 10, 2)	**10**	**2**	**0**	**5**	**10**	**5**	**2.0000**	**0.5000**	**4.0000**
	(10, 6, 10, 2)	10	2	2	4	10	6	2.0000	0.4000	5.0000
	(10, 7, 10, 2)	10	2	4	3	10	7	2.0000	0.3000	6.6667
	(10, 8, 10, 2)	10	2	6	2	10	8	2.0000	0.2000	10.0000
	(10, 9, 10, 2)	10	2	8	1	10	9	2.0000	0.1000	20.0000
Increasing N	(6, 5, 10, 2)	10	2	0	1	6	5	1.2000	0.1667	7.2000
	(7, 5, 10, 2)	10	2	0	2	7	5	1.4000	0.2857	4.9000
	(8, 5, 10, 2)	10	2	0	3	8	5	1.6000	0.3750	4.2667
	(9, 5, 10, 2)	10	2	0	4	9	5	1.8000	0.4444	4.0500
	(10, 5, 10, 2)	**10**	**2**	**0**	**5**	**10**	**5**	**2.0000**	**0.5000**	**4.0000**
	(11, 5, 10, 2)	10	2	0	6	11	5	2.2000	0.5455	4.0333
	(12, 5, 10, 2)	10	2	0	7	12	5	2.4000	0.5833	4.1143
	(13, 5, 10, 2)	10	2	0	8	13	5	2.6000	0.6154	4.2250
	(14, 5, 10, 2)	10	2	0	9	14	5	2.8000	0.6429	4.3556
Increasing L	(10, 5, 100, 20)	100	20	0	5	10	5	2.0000	0.5000	4.0000
	(10, 5, 200, 40)	200	40	0	5	10	5	2.0000	0.5000	4.0000
	(10, 5, 300, 60)	300	60	0	5	10	5	2.0000	0.5000	4.0000
	(10, 5, 400, 80)	400	80	0	5	10	5	2.0000	0.5000	4.0000
	(10, 5, 500, 100)	500	100	0	5	10	5	2.0000	0.5000	4.0000
Ex	(8, 3, 10, 4)	10	4	2	5	8	3	3.2000	0.6250	5.1200

Using Rabin's IDA requires the number of symbols per file (L), the number of symbols per Fragment (K), and the number of resultant file pieces (N) to be given. Table 17

shows that IDA can provide a wide range of security and cost optimized configurations. For a high security configuration a user might use the (10, 2, 10, 5) configuration which has a redundancy factor of 0.8. For a space efficient configuration, a user might use the (6, 5, 10, 2) configuration which has a resultant size factor of 1.2. For a balanced configuration, a user can use any configuration where $N = 2 \times K$, such as (10, 5, 10, 2) where its RSF/RF ratio is 4.0, the lowest for Rabin's IDA.

From a security point of view, Rabin's IDA achieves both confusion and diffusion as the matrix multiplication operation masks the original data, and spreads the effects of the bits to all N pieces of resultant files. An update to the original file will require the all symbols and file pieces to be recomputed and updated. As such, IDA does not support partial updates.

6.3.7 Reed-Solomon Codes

In 1960, Irving S. Reed and G. Solomon published "Polynomial Codes over Certain Finite Fields" describing a family of efficient, max distance separable error correction codes based upon polynomial construction and deconstruction [53] over a Finite Field. The family of codes is named Reed-Solomon (RS) codes after their inventors.

An (N, K) RS code exists over a Finite Field of $GF(2^M)$ where $N = 2^M - 1$. M is chosen to correspond with common bit-lengths such as Word (4), Byte (8), and 16, 32, and 64 for corresponding CPU architectures. For network applications, the traditional home field for RS codes, M is typically 8. For GF(16), M = 4. In all codes, K is chosen depending on the level of redundancy the designer wants in the code. When N – K is

even, then 2t = N – K, if it is odd, then 2t = N – K – 1. Simplifying these equations, $t = \lfloor (N - K) \div 2 \rfloor$.

An (N, K) RS code can self-repair up to t errors, or 2t erasures. RS codes can detect up to t error locations and correct them through its standard decoding process, however if the error locations are known ahead of time, then it can correct up to 2t errors directly.

An (N, K) RS code is constructed by forming the generator polynomial G(X) consisting of N – K factors, the roots of the polynomial are consecutive elements in the Finite Field, as shown in Equation 12. Some codes start with b = 0, while others start with b = 1.

Equation 12 – Reed-Solomon Codes Generator Polynomial

$$G(X) = (X + \alpha^{b})(X + \alpha^{b+1})\ldots(X + \alpha^{b+2t-1})$$

For example, the Generator Polynomial for the (15, 11) RS code with B = 0 is calculated as follows:

$$2t = 15 - 11 = 4, \text{ thus four factors in } G(X).$$

$$2t - 1 = 3$$

$$G(X) = (X + \alpha^{0})(X + \alpha^{1})(X + \alpha^{2})(X + \alpha^{3})$$

$$\text{Since, } \alpha = 2$$

$$G(X) = (X + 1)(X + 2)(X + 4)(X + 8) = X^{4} + 15X^{3} + 3X^{2} + X + 12$$

Substituting the coefficients with the field element values from Table 2, we get Equation 13 which is the generator polynomial for the (15, 11) RS code.

Equation 13 – Generator Polynomial for (15, 11) Reed-Solomon Code

$$G(X) = \alpha^0X^4 + \alpha^{12}X^3 + \alpha^4X^2 + \alpha^0X + \alpha^6$$

The encoding procedure for RS codes starts by dividing the file into a number of messages $M_i(X)$. Each message, denoted as M(X) for simplicity, is further divided into K information symbols each M bits long.

$$M(X) = M_{K-1}x^{K-1} + \dots M_1x + M_0$$

Each coefficient M_{K-1}, … M_1, M_0 is an M-bit message symbol corresponding to an element of $GF(2^M)$.

To form the encoded code word, multiple each message M(X) by X^{N-K}, then divide by G(X):

Equation 14 – Reed-Solomon Encoding Computation

$$\frac{M(X) \times X^{N-K}}{G(X)} = Q(X) + \frac{R(X)}{G(X)}$$

Division by G(X) produces quotient Q(X) and remainder R(X) polynomials, where R(X) is of degree up to N – K – 1.

The encoded code word T(X) is formed as the message bits joined with the remainder bits, shown in Figure 8. Computation they can be computed by shifting M(X) by X^{N-K} bits then adding R(X), as follows:

$$T(X) = M(X) \times X^{N-K} + R(X)$$
$$= M_{K-1}X^{N-1} + \cdots + M_0X^{N-K} + R_{N-K-1}X^{N-K-1} + \cdots + R_0$$

Figure 8 – Encoded Code Word for Reed-Solomon Codes

Rewriting Equation 14, we have the following identities:

$$M(X) \times X^{N-K} = G(X) \times Q(X) + R(X)$$

$$M(X) \times X^{N-K} + R(X) = G(X) \times Q(X) = T(X)$$

As such, T(X) is always divisible by G(X) without remainder. This is the condition checked by a system using RS coding to ensure a code word has no errors.

Consider the received code word as R(X) = T(X) + E(X), where E(X) represents any received errors. The error correction process follows a 5 step procedure [59]:

Table 18 – Reed Solomon Error Correction Process

Step	Process	Runtime
1	Calculate and find all Error Syndromes. Up to t syndromes can be found.	O(|F|)
2	Use the Berlekamp-Massey Algorithm to compute the error locator polynomials.	$O(\|F\|^2)$
3	Use Chien Search to find all the error locations. Specifically, find the roots of the error location polynomials L(X) over the GF field, which gives us the error locations.	O(|F|)
4	Use Forney's Formula to compute the error magnitudes at each error location, giving us E(X)	O(|F|)
5	Solve for T(X): T(X) = R(X) – E(X)	O(|F|)

To use RS codes in erasure correction mode, a checksum would have to be computed for every symbol, and then checked prior to the decoding process. The decoding process in this case first flags the symbols and locations which do not have a matching checksum, and then uses Forney's Formula in step 4 to compute the error magnitudes in order to reconstruct T(X).

In both error correction mode and erasure correction mode, M(X) is extracted from T(X) after checking that T(X) is correct. Extrapolating towards an entire file, the sequence of messages $M_i(X)$, once extracted individually from each RS encoded code word, would be combined in the correct sequence to reconstruct the original file. To store the code words on the Cloud, each symbol within a code word is grouped with the corresponding symbols sharing the same position in other code words, then it is sequenced in the same order as the set of messages $M_i(X)$, forming the output files pieces.

Out of N total symbols of each code word, K of them are information symbols, and N – K of them are redundancy symbols. This gives a resultant space of $N \times |F| \div K$, resultant space factor of $N \div K$. RS codes gives a redundancy factor of $t \div N$ if used in error correction mode, and alternatively it gives a redundancy factor of $2t \div N = (N - K) \div N$ if used in erasure correction mode. A computation key must be kept to store the Finite Field elements and their values. A reconstruction relation table must also be kept in the metadata to store the index and sequence of messages in relation to the original file.

Table 19 shows the redundancy performance of various RS codes for M = 2, 4, and 8. Using the code in erasure correction mode would yield double the redundancy factor compared to using the code in error correction mode. The resultant size of each code word is the same, although in erasure correction mode some extra data must be kept in the metadata file for the checksums. Overall the performance favors using RS codes in erasure correction mode. RS codes are the safest when K is minimal; this is evident regardless of what value of M is chosen. Whenever K = 1, the highest possible redundancy factor is achieved. However, space wise the most efficient codes exist

when $K = \lfloor N \div 2 \rfloor$, or half the number of symbols. These are the bolded rows in Table 19, which includes the (3, 1), (15, 7), and (255, 127) RS codes.

Table 19 – Reed-Solomon Codes and Redundancy Performances

								Error Correction Mode		Erasure Correction Mode	
Reed-Solomon Code (N, K)	**M**	**GF**	**N**	**K**	**R**	**T**	**RSF**	**RF**	**RSF/RF**	**RF**	**RSF/RF**
(15, 2)				2	13	6	7.5000	0.4000	18.7500	0.8000	9.3750
(15, 3)				3	12	6	5.0000	0.4000	12.5000	0.8000	6.2500
(15, 5)				5	10	5	3.0000	0.3333	9.0000	0.6667	4.5000
(15, 7)	4	16	15	**7**	**8**	**4**	**2.1429**	**0.2667**	**8.0357**	**0.5333**	**4.0179**
(15, 9)				9	6	3	1.6667	0.2000	8.3333	0.4000	4.1667
(15, 11)				11	4	2	1.3636	0.1333	10.2273	0.2667	5.1136
(15, 13)				13	2	1	1.1538	0.0667	17.3077	0.1333	8.6538
(255, 10)				10	245	122	25.5000	0.4784	53.2992	0.9569	26.6496
(255, 20)				20	235	117	12.7500	0.4588	27.7885	0.9176	13.8942
(255, 40)				40	215	107	6.3750	0.4196	15.1928	0.8392	7.5964
(255, 60)				60	195	97	4.2500	0.3804	11.1727	0.7608	5.5863
(255, 80)				80	175	87	3.1875	0.3412	9.3427	0.6824	4.6713
(255, 100)				100	155	77	2.5500	0.3020	8.4448	0.6039	4.2224
(255, 120)				120	135	67	2.1250	0.2627	8.0877	0.5255	4.0438
(255, 127)	8	256	255	**127**	**128**	**64**	**2.0079**	**0.2510**	**8.0001**	**0.5020**	**4.0001**
(255, 140)				140	115	57	1.8214	0.2235	8.1485	0.4471	4.0742
(255, 160)				160	95	47	1.5938	0.1843	8.6469	0.3686	4.3235
(255, 180)				180	75	37	1.4167	0.1451	9.7635	0.2902	4.8818
(255, 200)				200	55	27	1.2750	0.1059	12.0417	0.2118	6.0208
(255, 220)				220	35	17	1.1591	0.0667	17.3864	0.1333	8.6932
(255, 240)				240	15	7	1.0625	0.0275	38.7054	0.0549	19.3527
(255, 255)				255	0	0	1.0000	0.0000	Undefined	0.0000	Undefined
(255, 223)	8	256	255	223	32	16	1.1435	0.0627	18.2245	0.1255	9.1122
(255, 239)	8	256	255	239	16	8	1.0669	0.0314	34.0089	0.0627	17.0044
(255, 251)	8	256	255	251	4	2	1.0159	0.0078	129.5319	0.0157	64.7659

Interestingly, the most popular RS codes in use – the (255, 223), (255, 239), and (255, 251) codes, shown at the bottom in Table 19 – do not provide much redundancy nor is it very efficient at providing its respective level of redundancy. These codes are used in modern day satellite communications, optical disk storage encoding, and many other communications tasks.

6.3.8 Overall Comparison

All seven erasure codes are compared in this section for their redundancy performance, security properties, and algorithmic efficiencies in this section.

Table 20 – Best Redundancy Performance Erasure Code Configurations

Code Type	Most Efficient Configuration	Example Configuration	Resultant Space Factor (RSF)	Redundancy Factor (RF)	Risk (RSF/RF)
Simple Replication	(2, 1)		2.0000	0.5000	4.0000
Hamming Code	(7, 4)		1.7500	0.4286	4.0833
RAID-5	(3, 2)		1.5000	0.3333	4.5000
LDPC	Any (N, K, PT) code where N = 2 × K	(6, 3, 2)	2.0000	0.5000	4.0000
Shamir's Secret Sharing	(4, 2, 1)		4.0000	0.5000	8.0000
Rabin's IDA	Any (N, K, L, M) code where N = 2 × K	(10, 5, 10, 2)	2.0000	0.5000	4.0000
Reed-Solomon	Any (N, K) code where N = 2 × K - 1, large N improves minorly	(225, 127)	2.0079	0.5020	4.0001

Table 20 shows the configurations of each erasure code at their best redundancy performance. A code achieves its best redundancy performance when the ratio of RSF/RF is at its lowest. The ratio is essentially the cost divided by the gain; namely, the resultant space taken by the set of files after encoding divided by the safety afforded by the set of files represented by the factor of files pieces that could be completely lost. When this ratio is minimized, we achieve the best trade-off of space cost vs file safety gained. For all codes, it seems that their best performance is achieved when $N = 2 \times K$. This implies that configurations should produce the same number of redundancy file pieces as there are original file pieces.

It is not easy to find a single configuration that works for all codes, except for the most trivial of (2, 1) for which in all codes except Rabin's IDA they would carbon copy the original file once. To compare the codes' performances when set to a

meaningful equivalent configuration, we attempted to configure each code as close as possible to a (15, 11) configuration. The emphasis is on the 11 original file pieces, as the total number of pieces is the result of each code's encoding process. The results are shown in Table 21.

Table 21 – Redundancy Performances When Set As Close to (15, 11) Code Configuration

	Configuration	Code Properties	R	N	K	RSF	RF	RSF/RF
Simple Replication	(**15**, 1)	K is always 1	14	15	1	15.0000	0.9333	16.0714
Hamming Code	(**15**, **11**)	Bits to Recover 1 Other Bit = 7 Bit Overlap From any 2 Equations = 11 Bits Needed to Validate Another Bit = 4	4	15	11	1.3636	0.2667	5.1136
RAID-5	(12, **11**)	K is always N – 1	1	12	11	1.0909	0.0833	13.0909
LDPC	(22, **11**, 10)	PT = 10	11	22	11	2.0000	0.5000	4.0000
Shamir's Secret Sharing	(**15**, **11**, 10)	Poly Degree = 10	4	15	11	15.0000	0.2667	56.2500
	(22, **11**, 10)		11	22	11	22.0000	0.5000	44.0000
Rabin's IDA	(**15**, **11**, 22, 2)	Symbols Per File = 22, Fragments = 2, Extra Symbols = 0	4	15	11	1.3636	0.2667	5.1136
	(22, **11**, 22, 2)		11	22	11	2.0000	0.5000	4.0000
Reed- Solomon	(**15**, **11**)	M = 4, GF = 16, T = 2	4	15	11	1.3636	0.2667	5.1136

For Simple Replication, K is always 1 so the configuration was set to (15, 1). For RAID-5, K is always N – 1 so the configuration was set to (12, 11). For LDPC, since N is a result of the chosen PT and K values, the configuration was set to (22, 11, 10). The other four codes were able to configure for (15, 11) directly.

The most restrictive codes in terms of configuration flexibility are Simple Replication, Hamming Code, and RAID-5. The least restrictive codes are Shamir's Secret Sharing, Reed-Solomon, and Rabin's IDA. LDPC is in the middle in terms of flexibility.

Except for Shamir's Secret Sharing and Rabin's IDA, the configurations in Table 21 are the most efficient, or otherwise the only configurations, possible for each of the codes for this target. For these two codes, their most efficient configurations of (22, 11) are also shown in the table.

Table 21 shows that the two most efficient codes are LDPC and Rabin's IDA. Both of these codes achieve RSF of 2.0, RF of 0.5, and RSF/RF of 4.0 in their most efficient configurations of (22, 11, 10) and (22, 11, 22, 2) respectively. Rabin's IDA can also be configured with lower redundancy to save some disk space in the (15, 11, 22, 2) configuration. It can also be configured with higher redundancy if more security is desired. Its flexibility makes it the best performing code among the seven studied.

Reed-Solomon codes are slightly more restrictive in terms of the number of possible configurations compared to Rabin's IDA, however its performance closely matches IDA and in its theoretical upper bound it can also achieve the same performance as Rabin's IDA. However, Reed-Solomon does not have the Confusion Property so it is less secure on its own compared to Rabin's IDA.

The worst performing code is Shamir's Secret Sharing which does not seem to perform better than Simple Replication in these configurations. If Shamir's is configured in (15, 11, 10) it has the same resultant space factor as Simple Replication, however it yields lower redundancy factor of 0.2667 compared to Simple Replication's 0.5, which implies that it can tolerate less errors and file losses. If Shamir's was configured to match the redundancy factor of 0.5, it yields a resultant space factor of 22.0 compared to Simple Replication's 15.0, which implies it costs more space to provide the same level of redundancy.

Table 22 shows that of the codes studied only Shamir's Secret Sharing and Rabin's IDA have the Confusion Property. Both of them involve a more costly decoding processes requiring at minimum $O(|F|^2)$ time complexity for decoding time. Between these two, Rabin's IDA can achieve a lower resultant space factor, thus is more

efficient as it consumes less space to provide a similar level of redundancy as Shamir's Secret Sharing. The trade-off is that Rabin's IDA requires more memory space than Shamir's Secret Sharing during both encoding and decoding operations.

Table 22 – Erasure Code Properties and Redundancy Formulas

Code Type	Simple Replication	Hamming Code	RAID-5	LDPC	Shamir's Secret Sharing	Rabin's IDA	Reed-Solomon
Mathematical Principle	Copy and Paste	XOR of Related Bits	XOR of All Bits	XOR of Bits in Specific Arrangements	Polynomial Interpolation	Matrix Dot Product and Matrix Inversion	Polynomial Remainder Over Finite Field
File Reconstruction Threshold (FT)	1	K	K	K	K	K	K
File Piece Reconstruction Threshold (PT)	1	$2^{M-1}-1$	K	$\frac{K}{2} \le PT \le K-2$	K	K	$N-\left\lfloor\frac{N-K}{2}\right\rfloor$
Resultant Space (Bytes)	$N \times \lvert F\rvert$	$2^{M}-1$	$\lvert F\rvert + \frac{\lvert F\rvert}{K}$	$\frac{((K \text{ choose } PT)+K)\times\lvert F\rvert}{K}$	$N \times \lvert F\rvert$	$\frac{N\times M\times\lvert F\rvert}{L}$	$\frac{N\times\lvert F\rvert}{K}$
Resultant Space Factor	N	$\frac{2^{M}-1}{2^{M}-M-1}$	$\frac{K+1}{K}$	$\frac{(K \text{ choose } PT)+K}{K}$	N	$\frac{N\times M}{L}$	$\frac{N}{K}$
Redundancy Factor	$\frac{N-1}{N}$	$\frac{M}{2^{M}-M-1}$	$\frac{1}{K+1}$	$\frac{(K \text{ choose } PT)+K}{(K \text{ choose } PT)}$	$\frac{N-K}{N}$	$\frac{N-K}{N}$	$\frac{N}{T}$
Encoding Time	$O(\lvert F\rvert)$	$O(\lvert F\rvert)$	$O(\lvert F\rvert)$	$O(\lvert F\rvert)$	$O(\lvert F\rvert)$	$O(\lvert F\rvert^2)$	$O(\lvert F\rvert)$
Decoding Time	$O(1)$	$O(\lvert F\rvert)$	$O(\lvert F\rvert)$	$O(\lvert F\rvert)$	$O((K\times\lvert F\rvert)^2)$	$O(\lvert F\rvert^2)$	$O(\lvert F\rvert^2)$
Temporary Space for Encoding (Bytes)	0	~ 0	~ 0	$(N-K)\times K\times PT$	$N\times M$	$O\left(\lvert F\rvert + \frac{\lvert F\rvert\times M}{L}\right)$	$O(\lvert K\rvert)$
Temporary Space for Decoding (Bytes)	0	~ 0	~ 0	$\lvert F\rvert$	$O((K\times M)^2)$	$O\left(\lvert F\rvert + \frac{\lvert F\rvert\times M}{L}\right)$	$O(2\times\lvert N\rvert)$
Confusion Property	No	No	No	No	Yes	Yes	No
Diffusion Property	Yes	No	Yes	Yes	Yes	Yes	Yes
Partial Updates Supported	Yes	Yes	Yes	Yes	No	No	No
Metadata - Computation Key Required	No	No	No	Yes	Yes	Yes	Yes
Metadata - Reconstruction Relation Table Required	No	Yes	Yes	Yes	Yes	Yes	Yes
Notes		M = number of parity bits = (N – K)			M = number of parity bits	L = symbols per file; M = number of fragments	

Another interesting observation is that only the XOR based codes support partial updates. This is useful to users since it can be expected that they would use a Cloud storage system to incrementally backup daily or weekly changes to files. Not having to re-upload all file pieces makes the system efficient and can save costs.

The factors of Encoding Time, Decoding Time, Temporary Space for Encoding, Temporary Space for Decoding, Requirements for Metadata, and ability to support Partial Updates are useful for systems designers who are designing a redundant Cloud or distributed storage system. The security properties of Diffusion and Confusion are useful indicators of the algorithms' ability to combat targeted attacks which attempt to gain the information contained in the files. The Resultant Space Factor and Redundancy Factor are of utmost interest in this thesis as it directly affects the file security, resiliency, and economic costs of a system given particular choices of algorithms. From a practical perspective, a code having Confusion and Diffusion Properties are cryptographically stronger, as they have an equivalent capability of essentially performing encryption on the contents of the file in addition to their ability to split the file and add redundancy. For the other codes studied, one means for them to "gain" the Confusion and Diffusion Properties is to encrypt the file using well known encryption algorithms before, or after using the code to split the encrypted file into pieces.

Applying encryption before splitting is arguably stronger since it can be done at an operating system level with many choices of tools and algorithms, and it also safeguards the user's data against other conventional attacks directly on their devices. The storage system should still be in charge of ensuring that a file has been encrypted prior to any encoding operations. Since diffusion implies that every input

bit is involved in computing an encrypted output bit, any updates to the original file would result in a completely different encrypted file. The storage system would have to re-encode the new encrypted file and produce a new set of file pieces, regardless of the erasure code algorithm used. Applying encryption before splitting results in a system that cannot support partial updates; a crucial computation, network bandwidth, and cost saving system property. Despite this shortcoming, this approach has been applied in prior work by Hugo Krawczyk of IBM, published in his paper "Secret Sharing Made Short" in 1993 [60].

Applying encryption after splitting makes the file pieces strong and resilient towards attacks. In this mode, each file piece is encrypted separately after the encoding operation. The storage system would be in charge of encrypting and decrypting each file piece, which could be implemented by operating system level functions or external encryption software. Applying encryption after splitting preserves the partial update properties of the relevant erasure codes. However, each output file piece will have to be re-encrypted in its entirety. In both cases, it will add to the total encoding and decoding time.

All stream-based and block-based encryption algorithms, such as AES, DES, Blowfish, RC5, and IDEA have a linear time complexity relative to the size of the input file. However they need to be used in a proper mode of operation, such as Cipher Block Chaining, or Electronic Code Book modes to ensure the cryptographic security of the files. Other types of encryption algorithms exist with varied time complexities.

In conclusion, one of two types of system design is recommended from this Framework. In the first, the system should use a fast linear-time erasure code with

support for partial updates, and pair it with a suitable encryption algorithm where encryption is applied after the encoding step. From the codes studied, LDPC would be most suitable due to its configuration flexibility. This design allows for strong file crypto security from the encryption process, and strong resiliency against storage provider mistakes due to the erasure coding, and high cost efficiencies from its ability to support partial updates. In the second design, the system should use an erasure code which has both the Confusion and Diffusion properties so that it is not dependent upon any encryption systems or software to provide cryptographic security to files. From the codes studied, Rabin's IDA erasure code is suggested due to its higher resultant file size efficiencies compared to Shamir's Secret Sharing.

6.4 Handling Metadata

In Krawczyk's paper [60], he describes a space efficient secret sharing scheme combining Rabin's IDA algorithm with a secure encryption scheme and a perfect secret sharing scheme together to form a cryptographically strong secure storage system. Specifically, the three subsystems work in conjunction as follows:

Encrypt an original file with a random encryption key P, resulting in the encrypted file E. Split the encrypted file E using IDA into fragments E_1, E_2 … E_N. Use Shamir's Secret Sharing to generate N shares for the key P denoted as P_1, P_2 … P_N. For each storage repository i = 1, 2 … N, store E_i and P_i as a pair in that repository. Both IDA and Shamir's is set to the same threshold configuration, such that only K pairs, $K < N$ are necessary to recover the encrypted file E and the encryption key P.

In Section 6.3.8 we've concluded that Rabin's IDA algorithm itself exhibits both the Confusion and Diffusion properties, thus applying encryption prior to the use of IDA

on the file adds a layer of cryptographically security. Using Shamir's to split the encryption key is an efficient application of the code, since the size of the encryption key is usually significantly smaller than the size of the file. The space inefficiency of Shamir's Secret Sharing has much less impact in this set up compared to the gain of using its perfect cryptographic security property.

In Krawczyk's system, the encryption key and the encoding matrix used by IDA form the metadata used by the system. It is assumed that the encoding matrix is constant in their system (for example, always using a Vandermonde matrix), thus it does not need to be separately stored. However, since each fragment and key pair has a corresponding index, the index information must still be stored locally on the user's computer. This presents one potential weakness of Krawczyk's system.

In our Framework, we want to cryptographically secure any and all metadata used. The metadata would also be stored in a file. Directly adopting Krawczyk's approach of using Shamir's Secret Sharing scheme to secure and generate shares of the metadata file is a plausible solution. However, considering that the Framework needs to track these shares and their corresponding storage locations, the shares must be tracked by yet another index or metadata file. The problem thus propagates forward and remains unsolved.

Peer-to-Peer (P2P) storage systems present a solution to this problem of securing the metadata files while allowing them to tolerate against equipment failure, outages, or errors. Like P2P file sharing systems such as the Torrent networks, the principle ideas behind P2P storage systems is that a file would be stored in multiple peer locations in a peer-to-peer network.

The Framework encrypts the metadata with an encryption key which is chosen by the user, for example a SHA hash of a chosen password that the user can easily remember. Then, the encrypted metadata is distributed in a secure and private P2P network consisting of only the user's devices and computers. Encrypted metadata files will duplicate and propagate through this network incrementally to every device in the network. To access a metadata file, the user simply has to enter his or her password and the Framework will decrypt the file accordingly. It is important that any implementation of the Framework never stores the user's password or the hash of the user's password in any temporary or permanent storage. To know that a file has been decrypted properly the system adds flag check bits when creating an unencrypted metadata file, so that when the file is properly decrypted, those bits and the file will pass corresponding tests. The only way for the system to decode properly is if the user enters a correct password.

Secure P2P networks utilize encrypted communications links between every pair of devices, and can only be joined by authenticated and authorized user(s). If a user shares a file to another user, they will join that particular file's P2P network and obtain a copy of the encrypted metadata file. Every message in a secure P2P network is checked for its authenticity, thus any unauthorized messages being sent through that network would simply be ignored by the peers in the network. Other forms of secure P2P networks and secure P2P file systems exist, an example is MIT's Ivy P2P file system which is discussed in Section 7.3 [61].

It is assumed that the user will always have at least two devices online at any time, which is very common today due to the popular use of smart mobile phones. Devices such as smart phones which have limited processing power and storage capacities

only need to be involved in the metadata portion of the Framework, to save resources. They could also encode and decode files if the user desires; the Framework does not enforce any limitations in this aspect.

6.5 Cloud Storage Selection

The choice of which Cloud storage service provider to use depends on four factors of economic pricing, service provider system security, reliability, and geographical location related risks. This section examines these four factors in detail and concludes with a prioritization of the factors.

6.5.1 Economic Pricing Factor

From an end user's perspective, the internet service provider costs are often a fixed cost since service contract terms range from one year upwards to multiple years. The Framework, and any implemented systems, can only control the amount of data sent or retrieved from the internet in order to reduce the internet transmission costs shown earlier in Section 3.3.1. With enough pricing and usage information known, the Framework could suggest to users which other internet service providers would be more cost effective to establish contracts with. This function would have more impact towards business organizations than home users, since it would be exhaustive to tabulate all other uses of the internet by a home user.

For any new files being uploaded to the Cloud for the first time, the Framework would pick a set of storage providers which has the lowest current storage costs. The number of providers can be customized according to user preferences and security requirements. Generally, the Framework would prefer taking full advantage of any free usage tiers from storage providers before incurring storage costs. This would

need to be balanced with the security requirements. Using this approach, the Framework would minimize all short term costs as file pieces newly uploaded to the Cloud would incur the lowest possible storage costs given the present pricing. In terms of internet transmission costs, only a forward upload cost would be incurred.

In terms of long term storage costs, it can be expected that the storage providers would compete in terms of pricing and change according to the economic laws of demand, supply, and competition. In Section 5.4, point 7, it was mentioned that a user can face a dilemma of either paying higher long term costs by staying with an expensive provider, or pay an expensive transfer fee to move their files to the less expensive of the two Cloud storage providers for long term savings. The Framework lets the user reconstruct the file pieces stored on the expensive provider locally on their computer, and then upload these directly to the less expensive provider. The user doesn't have to pay a download fee or an outbound data fee in order to take advantage of the savings.

The Framework can model the re-upload costs as a fixed one-time fee which can be amortized over a period of time whereby the effective cost of storing the file in a cheaper provider is the same as staying with their current provider. We denote the upload fee X, the current provider's monthly storage price in $/GB as A, the new provider's monthly storage price as B, and the amortization period T in months. A simple equation can compute the value of T:

$$A \times T = X + B \times T$$

$$(A - B) \times T = X$$

Equation 15 – Storage Cost Amortization Period with Single Upload

$$T = \frac{X}{(A - B)}$$

Equation 15 assumes that the necessary file pieces for reconstructing a high cost file piece are already available locally on the user's computer. If these pieces need to be downloaded, a slightly modified equation can be used to compute T. Let the number of pieces that need to be downloaded be K, and the download cost per piece be Y. T can be computed as follows:

$$A \times T = X + K \times Y + B \times T$$

$$(A - B) \times T = X + K \times Y$$

Equation 16 – Storage Cost Amortization Period with Downloads

$$T = \frac{X + K \times Y}{(A - B)}$$

Anytime a pricing change occurs with any storage provider, the Framework can ask the user whether they expect to keep the file on the Cloud for longer than T months. If yes, the Framework can perform the necessary recomputation and upload tasks to move the relevant file piece(s) to a cheaper storage provider. Otherwise, the Framework will simply keep the file pieces in the existing set of providers until the user issues a delete command.

Using Equation 15 and Equation 16, the Framework can minimize short term and long term storage costs as well as data transmission costs.

6.5.2 Service Provider System Security and Reliability Factor

To deduce how secure and reliable a service provider is, the Framework can use data mining and web crawling techniques to find, track, and count the number of relevant security breaches, resolutions, and service outage events from trusted news websites to assign scores to each service provider. Generally, the more security breaches and service outages, the lower the score. The score would remain low until the issues have been resolved. Once scores are obtained, the Framework can select a subset of service providers who have the highest scores for this factor.

The Framework cannot take a direct approach to test each provider's systems for their security and reliability. Such tests can be viewed as malicious attacks, and can cause legal and financial liability for the user. Although data mining and web crawling technologies are interesting topics, they are beyond the scope of this thesis. An appropriate data mining implementation would be necessary in the Framework to be able to deduce the scores and track the service providers for this factor. However, the use of erasure codes does allow the Framework to tolerate some losses of file pieces.

6.5.3 Service Provider Geographical Location Factor

To deduce where are the geographical locations of the major data centers of each service provider, a data mining approach can be taken as well to look for information sources and data that indicate the locations. More importantly, the Framework should also obtain information about the world's geography for risky geographical zones such as areas that are much more susceptible to earth quakes, hurricanes, volcano eruptions, and natural disasters.

From these two sets of data, the Framework can construct a topological map to select a set of service providers whom have data centers in geographical locations with very low probabilities of incurring a natural disaster. The Framework would also prioritize spreading out the data physically across the globe as much as possible.

6.5.4 Prioritizing the Factors

The overall problem of choosing a set of Cloud storage providers could be thought of as an optimization problem of 1) minimizing storage and transmission costs, 2) finding and optimally moving data pieces to the most secure and reliable providers, and 3) distribute the file pieces to as sparse of geographical locations as possible, all at the same time.

Erasure codes increases the total file size by the Resultant Size Factor, previously analyzed throughout Chapter 6. Thus, relative to simply uploading files to a Cloud, the Framework adds both transmission and storage costs. The Economic Pricing Factor guides the Framework towards minimizing total costs.

The use of erasure codes by the Framework allows the complete loss and destruction of some file pieces. The Security and Reliability Factors guide the Framework towards more secure storage providers, thus reducing the probability of file pieces being lost or destroyed due to service provider mistakes, outages, and vulnerabilities.

Lastly, the Geographical Location factor guides the Framework to reduce the probability of file pieces being lost due to natural disasters, or being the subject of unnecessary data privacy intrusion due to local laws and customs.

The main security benefits of the framework are provided by the use of the erasure codes, and the Security and Reliability Factor of this section would only reduce the risks by selecting record-wise more secure providers. If the selection criterion for providers was purely based on security alone, and costs were unimportant, then the framework can directly use the Security and Reliability Factor in its decision making to select the lowest risk providers. Otherwise, the selection problem could be modeled as a monetary cost minimization problem. In this case, the prioritization order is as follows:

1) The Economic Pricing Factor is the most important since minimizing costs must be part of the Framework to convince users and organizations to adopt this approach. Users and organizations can rationally accept the trade-off of an increased storage space requirement and related costs for the security and reliability benefits of this Framework. Knowing that the Framework will actively try to minimize costs will add to its value, and potentially increase the rate of adoption and use of the system.
2) The Security and Reliability Factor is the second most important. Even though the Framework tolerates some losses, any loss of file pieces will involve computing the lost pieces and uploading them to another storage provider. Minimizing the probability of file pieces being lost also directly minimizes total costs. Ultimately, the security and reliability risks are controllable factors that a storage provider will constantly work to improve.
3) The Geographical Location Factor is the least important since natural disasters are not controllable. We have mentioned previously in Section 3.1.5 that privacy

intrusion issues ultimately require political dialogue and new laws to be formed across geopolitical boundaries.

6.6 CHAPTER SUMMARY

This chapter presented the design of a system Framework which solves the Cloud storage problem. The Framework is a template design which software engineers and security system designers can take and further refine into a detailed implementation. The four major components of the Framework are the erasure code transformation system, the metadata handling system, the Cloud storage management system, and the encryption system. In turn, they are responsible for transforming the files into pieces, recording and synchronizing metadata files across the user's devices, transferring the file pieces to and from the Cloud, and encrypting the files and file pieces throughout the Framework.

We presented the set of metrics used to analyze erasure codes for their security properties and computational efficiencies, which interested researchers can apply to other erasure codes to determine their feasibility and relative performances. We then presented the detailed analysis of seven erasure codes using the metrics, along with a comparative analysis at the end. We concluded that for erasure codes which do not have the Confusion property, it would be best to pair those codes with an encryption system to secure the data within the files. We also showed that codes which have the Confusion property have encoding and decoding operations which are higher in time complexity than the codes which do not have the Confusion property. The best performing algorithms were LDPC, which does not have the Confusion property, and Rabin's IDA, which does have the Confusion property.

We further analyzed the problem of securing the metadata files, showing how certain approaches in to this problem requires metadata files to protect metadata files in an essentially never ending chain. We proposed to encrypt the metadata file, and replicate it across a peer-to-peer network to the user's devices to provide redundancy.

Finally, we analyzed the problem of how to select a subset of Cloud storage providers from a master set. We defined three factors of economic pricing, service provider security and reliability, and service provider geography. We stated that if only security was the deciding factor, then the framework can use the security and reliability factor alone to select the least risky providers. Otherwise the factors are prioritized in the order presented, to let the Framework minimize the financial costs of using Cloud storage.

The next chapter compares and relates the approach and Framework to existing remote storage paradigms, and analyzes the existing paradigms for their vulnerabilities as well as strengths.

Chapter 7: Comparison to Existing Storage Paradigms

The Framework shares a number of similarities to existing methodologies and architectures with respect to the overall problem of storing data remotely in a secure and reliable fashion. It also has a number of key differences. This chapter examines the similarities and differences with three methodologies, namely a traditional Cloud storage system, a distributed file system, and a peer-to-peer file system.

7.1 Traditional Cloud Storage Architecture

In current Cloud storage systems, a user authenticates to a service provider's systems to establish a secure internet connection to the service. A user's files are securely sent to, and retrieved from the storage provider's servers through the internet connection. Depending on the storage provider, various redundancy and encryption algorithms are applied on the files to secure and safely store the file on the Cloud. A visualization of this model is shown in Figure 9 below.

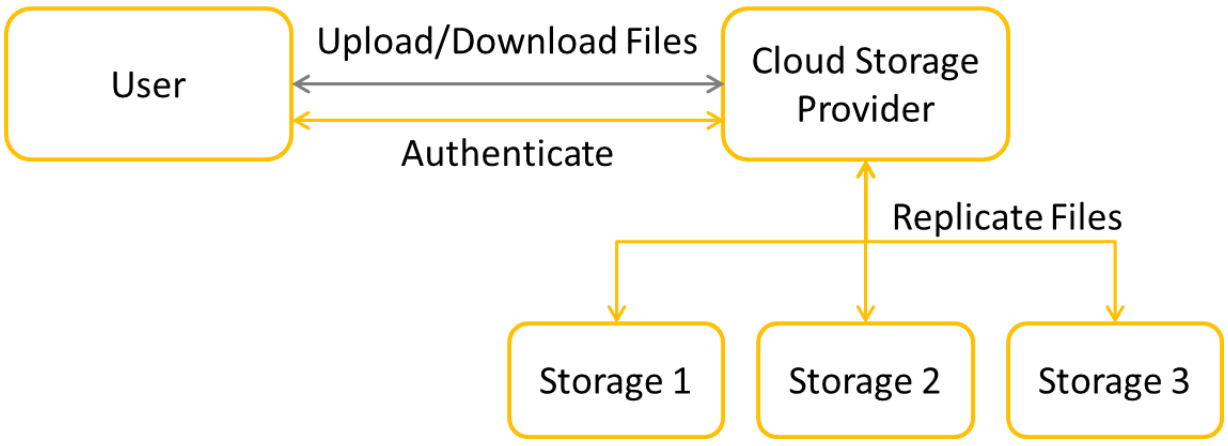

Figure 9 – Traditional Cloud Storage Architecture

This architecture is most vulnerable at the authentication mechanism. The architecture is blind to who is accessing a user's data when the authentication mechanism is compromised. Historically, the vast majority of attacks on Cloud storage providers have aimed at compromising the authentication mechanism [1] [2] [19] [30].

A secondary source of vulnerability has been the connections between the storage servers and the request handling servers within the Cloud storage provider's internal networks. In many data centers around the world these connections are not encrypted as the design of the data center's internal networks desired high efficiency, high transmission speed, and low latency. Encrypting connections within data center networks adds latency due to the computational time needed for encryption and decryption operations, reduces the speed as such secure connections require set up and tear down procedures, and reduces the efficiency as switch buffers and end nodes must queue data for longer periods of time waiting for these operations and procedures to finish in sequence. These types of unencrypted connections avail the system to attacks whereby the attacker wiretaps the connections and listens in on any and all data transmitted through the wire. As the storage nodes don't necessarily need to be within the same buildings or even the same city, external physical access to these connections becomes easy for attackers. Although as such vulnerabilities are discovered, companies work hard to amend them. For example, high speed encryption hardware systems are being designed by companies to secure transmission links, claiming performances at line rates of up to 10Gbps [62].

Some Cloud storage providers only utilize replication algorithms to backup data, while others employ erasure code algorithms. Using simply replication is weaker

from a security point of view as the data is readily available to be read by an attacker as long as they gain access to the physical disks storing the data, however it is more efficient computationally as any requests for data can be serviced without the need to reconstruct or recompute the data.

Some of the Cloud storage providers only replicate what is already available on a local storage space. These providers include Microsoft One Drive, Dropbox, and Google Drive when configured to synchronize with local files. When the local version is changed the system updates the version stored on the Cloud correspondingly, but when the local version is removed the Cloud version is also removed. Other storage providers act more alike a remote storage server where files could be uploaded and downloaded independent of local copies. These include Amazon's S3, Google Drive when not configured to synchronize with local files. Users can synchronize the local and Cloud versions through the system when desired.

The Framework borrows two important ideas from traditional Cloud storage architectures, firstly the use of erasure code algorithms, and secondly to use secured internet connections throughout all communication links. We've shown in Section 5.4 how the Framework resolves the problems faced by traditional Cloud storage systems.

7.2 Distributed File Systems and Architecture

In distributed file systems, a user connects to a remote storage server through, usually, a TCP/IP network to access their remote files. Most standard operating systems file operations such as copy, delete, create, move, open, close, and write

apply to these systems as well, and the user manages their files and folders on the remote server using the same user interfaces as they do on their own local computer.

A popular distributed file system is the Network File System (NFS) invented by Sun Microsystems in the 1980s [63]. In Linux and UNIX based systems, NFS folders could be accessed by first using the mount command to connect to the folder. A more recent and advanced distributed file system is Red Hat's GlusterFS, where the file system scales across many storage servers collectively considered as a virtual storage pool [64].

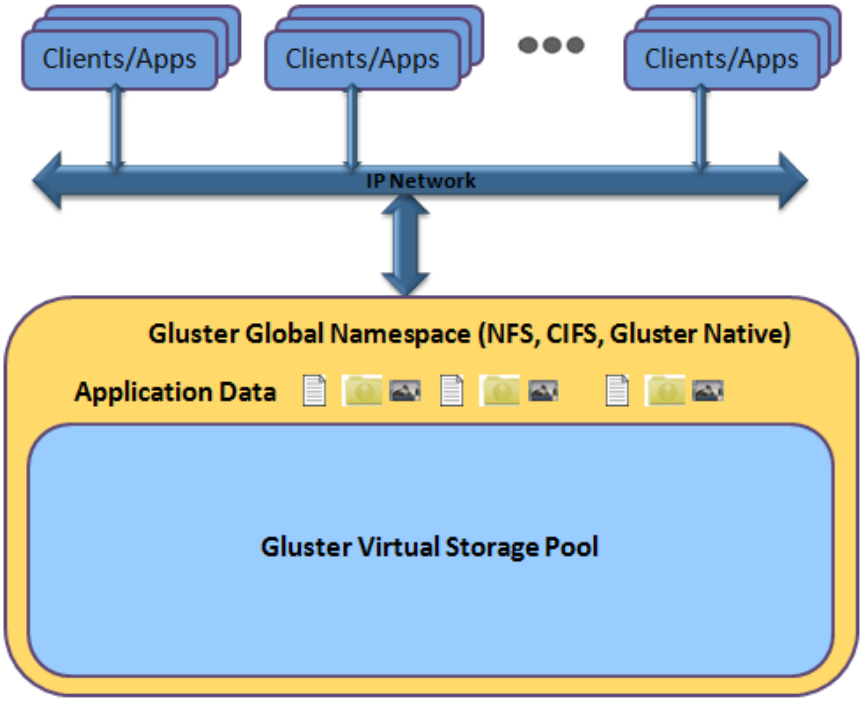

Figure 10 – GlusterFS Distributed File System Storage Architecture [64]

The first Cloud storage systems were built upon distributed file systems servers, which makes the two very similar. However, users have direct access to their folders and files in the remote storage server, which allows them to issue operating systems level commands to control their files. If a user issues a delete command by accident

to their remote files, it is unlikely that there would be any way to recover the file unless the distributed file system had versioning systems in place. In Cloud storage systems these types of operations are interpreted and carried through the software system, which allows Cloud vendors to always create backups and versions to each and every operation on any file. Part of the work of utilizing a distributed file system is in the administration and management of all of the remote folders and drive partitions. The administrator of the system has control over the physical location where each user's folders reside, which by extension implies the physical locations of all users' data. In Cloud systems, due to the nature and size of the data encountered, often the specific physical location is determined by load and space balancing algorithms. The remote storage server usually exists within a local area network of the user's home or corporate organization, rather than through the internet. Generally this means the user has control on the physical hardware of the storage servers, where as in the Cloud the users don't have control over the hardware. If users wanted to utilize distributed file systems, they would also have to bear the costs of the equipment, maintenance, and management of the remote file servers they use.

The Framework borrows the idea of carrying out file operations through a software system to allow for graceful recoveries from user mistakes, but at the same time strongly incorporates the notion of having much stronger control of the data, and where the data physically resides. Since the Framework ultimately uses Cloud storage providers to store users' data, it can save the user from having to invest in storage equipment.

7.3 Peer-to-Peer File Systems and Architecture

Peer-to-peer file systems (P2P FS) originated as a research and experimental file system platform aimed to take advantage of the decentralized control and scalability advantages of peer-to-peer networks. One of the most prominent systems is MIT's Ivy P2PFS [61]. Ivy is built on top of another tool called Chord [65]. Chord manages the coordination, and search of peers within the P2P network. Specifically, it assigns all participating peers into a Galois Field identifier circle, where all peers clockwise from the current peer is its successor peers. Each peer maintains a fixed table of immediate successor peers, which are indexed according to an interval range on the Galois Field. An example of this is shown below in Figure 11 for 3 peers with IDs of node 0, 1, and 3 respectively. Chord requires all peers in the P2P network to conform to this ordering specification, and whenever a peer joins or disconnects from the network, all other peers will update their successor tables. Peers are assigned into a node position based upon learning about information of one of the nodes in the ring through an external mechanism, such as from a P2P tracker server. Since every node in the ring knows its successors, a search for a peer within the network will simply traverse sufficient number of successor tables until it finds the peer in need. At a higher abstraction level, the location for a piece of data within the network is associated with a key which is mapped to the peer containing that data through a hash table that indexes the entire circle. Searches for data can simply be performed by searching using the corresponding key.

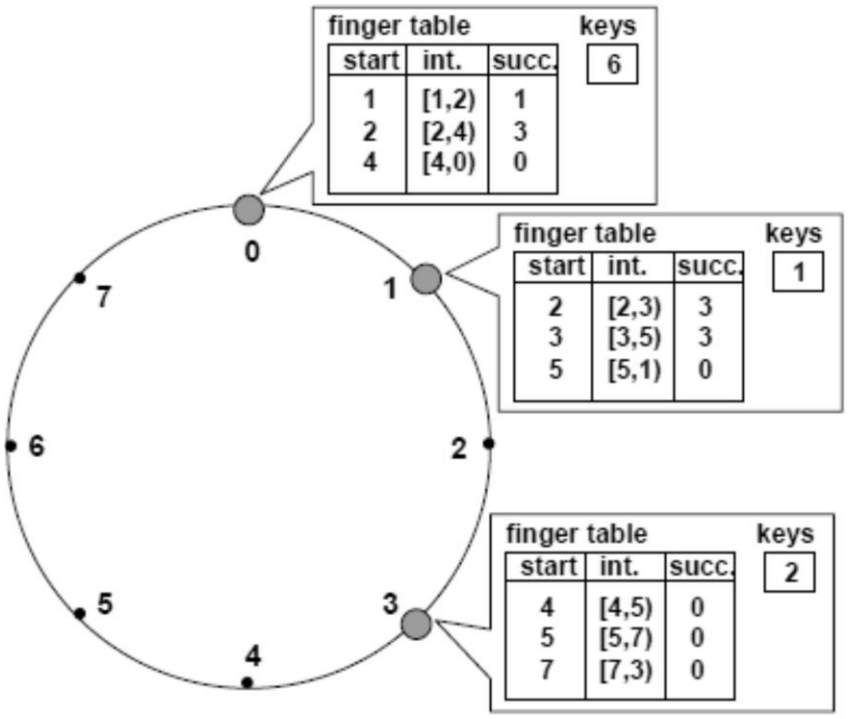

Figure 11 – Chord P2P Ring [65]

Ivy is a log based file system in that every file related operation is written into log files. To read a file, the system must consult all logs for all operations related to the file, and then compute the results to generate the output file. To write or change a file, the system writes only to the log file residing with the user making the changes. Within this overall operation scheme, it uses the mechanisms of Chord to distribute the replicas of the logs to a keyed peer node and its successor nodes. In the case of a user performing a modification operation on a file, its updated log is replicated to the user's successor peers. Ivy implements a number of other subsystems to ensure consistency of the logs, to resist attacks from outside the P2P network using encryption, and resist attacks within the P2P network through trust based log retrieval. In case of data corruption, Ivy also has tools that plausibly guess at the missing data blocks to assist in data recovery.

The premise of P2P file systems is that the decentralized control and access, along with sufficient numbers of participants in each network will be able to sustain a reasonable availability for any data stored in the network, and afford tolerance to equipment failures and outages through application of the Off-site Data Protection, and Replication principles. Compared to centralized storage systems like Cloud and its underlying distributed file systems, the probability of each node in the network to be offline is much higher, which makes P2P networks require much more replication than centralized systems. However, P2P networks such as Ivy are much more resistant to attacks from outside and inside the network, and the decentralized control makes it harder for an attacker to know where to begin attacking within a P2P network compared to centralized systems. The Ivy system also requires significant file-read overhead computation, since it has to retrieve all trusted logs through the network and compute the actual file from these logs.

The Framework takes advantages of the ideas behind Ivy to distribute the metadata among a set of peers in a P2P network, each peer being a mobile device or computer that the user owns. While there is a relatively higher cost in retrieving and reading metadata, the metadata is expected to be small relative to the size of the actual files so the computation time is minimized. Changes to the metadata can propagate through the network to the other peers incrementally as each device connects and disconnects from the network. Alternatively, the system can adopt the approach of using Ivy's read-all, write-local log system to store the metadata.

However, Ivy's file read cost overhead is too high for use for the actual data that a user would want to store in any off-site location. The Framework utilizes erasure code algorithms from the other storage systems in order to avoid this problem.

7.4 CHAPTER SUMMARY

This chapter analyzed the similarities and differences between the approach and Framework to existing remote storage paradigms, and analyzed the existing paradigms for their vulnerabilities as well as strengths. The Framework borrows the ideas of using erasure codes and secure internet connections from traditional Cloud storage architectures. We've shown in previous chapters how the approach is superior to a traditional Cloud storage architecture, however it worth keeping in mind that efforts by a service provider to improve their services under the traditional architecture will benefit a user using a system implemented from our Framework.

The distributed file system paradigm limits the user to essentially using network attached storage devices to back up their data, which involves heavy upfront costs, management and configuration work, and ultimately isn't as secure as a true off-site back up that is placed far away from the user's computing environment. However, distributed file systems allow users to have strong control over their files unlike traditional Cloud storage. The Framework borrows the idea of using a software system to perform file operations instead using operating system level commands in distributed file system architectures, in order to facilitate graceful recoveries from user mistakes. The Framework also borrows the idea of enforcing stronger control over the physical location of where data resides from distributed file systems.

Finally, the peer-to-peer file system paradigm offers an interesting but high computational overhead cost means to store data among a set of peers in a P2P network. While the notion of decentralized control helps to add to the work required of attackers, there are many security challenges and vulnerabilities of using P2P networks as the backbone for a file system. Many of these challenges were

confronted by the research team at MIT in their course of designing the Ivy P2P file system. The Framework adopts the use of a simple peer-to-peer network to distribute the metadata files generated to mobile devices and computer that the user owns, to secure the metadata files and to add redundancy to the metadata so that it can tolerate outages and failures.

The next chapter draws the final conclusions and discusses future works of this research problem.

Chapter 8: Conclusion and Future Work

In an ever increasingly online and interconnected world where we are expecting exponential growths of the amount of data being generated, stored, and processed on the internet, it is of vital importance to ensure the safety, security, reliability, and privacy of any personal information stored online.

The rise of Cloud Computing has given internet users a host of freedoms never enjoyed before, but comes with great risks for both users and the companies operating the Clouds. The analysis and Framework presented in this thesis showed, refined, and applied an effective approach based on the use of erasure codes. The Framework adds redundancy and cryptographic security to protect a user's personal data. This helps users mitigate the risks of using Cloud storage whilst reaping and enhancing its benefits. Along the way, a set of erasure code analysis metrics was presented, which can be used to conduct further research of other erasure codes for their applicability to this problem. The economic constraints and problems of using Cloud storage was also analyzed, resulting in a set of prioritized factors to help users and storage systems select Cloud storage providers.

The in-depth and comparative analysis of seven erasure codes showed that there's a corresponding computational complexity trade-off where codes with better data security incur higher computational time complexities. Where codes shared the same mathematical principles, the allowed arrangements and configurations decided the efficiencies of each code. From the seven codes analyzed, two winners were chosen

due to their configuration flexibility and inherent security and efficiency properties. LDPC was chosen because it was fast and offered many configurations. Similarly, Rabin's IDA was chosen because of its computational security and configuration flexibility. Ultimately, the choice of using a particular erasure code will depend on the security and efficiency preferences or needs of the user.

The knowledge, analysis, and the Framework presented in this thesis enables interested researchers and software professionals to perform detailed design and implementation of an actual software system to help secure user data on the Cloud. The benefits would be tremendous to end users.

As a research problem, the use of erasure codes is also a trade-off in that users must accept an increase in total storage costs in order to solve the Cloud storage problems. An even more ideal solution would find ways of minimizing or eliminating this increase. This is an open research challenge which might involve experimental research in erasure code designs, or the study and design of other kinds of data security systems.

The author hopes that the knowledge collected and analyzed in this thesis would be synthesized further into effective educational materials to help inform the public about Cloud Computing, and to guide software professionals on the right path towards developing secure data driven systems for the Cloud.

REFERENCES

[1] Sean Ludwig. (2011, June) venturebeat.com. [Online]. http://venturebeat.com/2011/06/21/dropbox-files-left-unprotected-for-four-hours-due-to-software-bug

[2] Andrew R Hickey. (2011, October) Researchers Uncover 'Massive Security Flaws' In Amazon Cloud. [Online]. http://www.crn.com/news/cloud/231901911/researchers-uncover-massive-security-flaws-in-amazon-cloud.htm

[3] Hussam Abu-Libdeh, Lonnie Princehouse, and Hakim Weatherspoon, "RACS: A Case for Cloud Storage Diversity," *SOCC 2010: ACM Symposium on Cloud Computing*, June 2010. [Online]. http://pubs.0xff.ps/papers/racs-socc.pdf

[4] James Steddum. (2013, July) A Brief History of Cloud Computing. [Online]. http://blog.softlayer.com/2013/virtual-magic-the-cloud

[5] Simson L. Garfinkel, *Architects of the Information Society: 35 Years of the Laboratory for Computer Science at MIT*, 1st ed. Boston, MA, USA: MIT Press, 1999. [Online]. http://books.google.ca/books?id=Fc7dkLGLKrcC&lpg=PA1&ots=J8GDnSVXbN&pg=PA1#v=onepage&q&f=false

[6] Jim Elliott. (2004, August) The Evolution of IBM Mainframes and VM. [Online]. http://www.linuxvm.org/Present/SHARE103/S9140jea.pdf

[7] Amazon.com, Inc. (2006) Amazon Simple Storage Service (Amazon S3). [Online]. https://web.archive.org/web/20061208012150/http://www.amazon.com/S3-AWS-home-page-Money/b?ie=UTF8&node=16427261&me=A36L942TSJ2AJA

[8] Claude E. Shannon, "Communication Theory of Secrecy Systems," *Bell System Technical Journal*, vol. 28, pp. 656-715, October 1949. [Online]. http://netlab.cs.ucla.edu/wiki/files/shannon1949.pdf

[9] Royal Canadian Mounted Police. (2013, January) Identity Theft and Identity Fraud. [Online]. http://www.rcmp-grc.gc.ca/scams-fraudes/id-theft-vol-eng.htm

[10] Latanya Sweeney, "Simple Demographics Often Identify People Uniquely," Carnegie Mellon University, Pittsburgh, 2000. [Online]. http://dataprivacylab.org/projects/identifiability/paper1.pdf

[11] Government of Canada, "Personal Information Protection and Electronics Documents Act, S.C. 2000, c.5," Ottawa, Law 2014. [Online]. http://laws-lois.justice.gc.ca/PDF/P-8.6.pdf

[12] Office of the Privacy Commissioner of Canada, "Your Guide to PIPEDA," Ottawa, 2009. [Online]. https://www.priv.gc.ca/information/02_05_d_08_e.pdf

[13] Jennifer Stoddart. (2004, April) An overview of Canada's new private sector privacy law - The Personal Information Protection and Electronic Documents Act. [Online]. https://www.priv.gc.ca/media/sp-d/2004/vs/vs_sp-d_040331_e.asp

[14] Jennifer Stoddart. (2007, August) Letter to Mr. David C. Drummond, Senior Vice-President, Corporate Development and Chief Legal Officer, Google, regarding 3D online mapping technology. [Online]. https://www.priv.gc.ca/media/let/let_070911_01_e.asp

[15] CBC News. (2007, September) Street View will comply with Canada's privacy laws: Google. [Online]. http://www.cbc.ca/news/technology/street-view-will-comply-with-canada-s-privacy-laws-google-1.674180

[16] Office of the Information Commissioner, Queensland. (2013, April) Drones – collection, storage and security of personal information. [Online]. http://www.oic.qld.gov.au/__data/assets/pdf_file/0009/17793/guideline-drones-collection-storage-and-security-of-personal-information.pdf

[17] 104th Congress of the U.S. Government , "HEALTH INSURANCE PORTABILITY AND ACCOUNTABILITY ACT of 1996," U.S. Government, Public Law 1996. [Online]. http://www.gpo.gov/fdsys/pkg/PLAW-104publ191/html/PLAW-104publ191.htm

[18] Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf, "DeepFace: Closing the Gap to Human-Level Performance in Face Verification," in *Conference on Computer Vision and Pattern Recognition (CVPR 2014)*, Columbus, OH, 2014. [Online]. https://www.facebook.com/publications/546316888800776/

[19] Financial Times. (2013, June) Digital hunter-gatherers. [Online]. http://www.ft.com/intl/cms/s/0/f840dbc0-d34f-11e2-b3ff-00144feab7de.html

[20] Facebook. (2013) Quarterly Earnings Slides Q4 2013. [Online]. http://files.shareholder.com/downloads/AMDA-NJ5DZ/3009267242x0x721748/be75c513-b84a-486d-a838-25cdc79c6a16/FB_Q413EarningsSlidesFINAL.pdf

[21] Facebook. (2014, January) Facebook Reports Fourth Quarter and Full Year 2013 Results. [Online]. http://files.shareholder.com/downloads/AMDA-NJ5DZ/3009264082x0x721811/f028299e-a5b9-4ed5-9a2d-e3f0923ef261/FacebookReportsFourthQuarterAndFullYear2013Results.pdf

[22] Sony Corporation. (2011, May) Sony Online Entertainment Announces Theft of Data from Its Systems. [Online]. https://www.soe.com/securityupdate/pressrelease.vm

[23] Kazuo Hirai, Sony Corporation. (2011, May) Sony's Response to the U.S. House of Representatives (on Flickr). [Online]. https://www.flickr.com/photos/playstationblog/5687532568/in/set-72157626521862165/

[24] Sony Online Entertainment. (2011, July) SOE Game Pass Plans. [Online]. https://web.archive.org/web/20110702094956/http://www.soe.com/gamepass/

[25] (2011, May) PlayStation Network breach will cost Sony $171M. [Online]. http://www.theregister.co.uk/2011/05/24/sony_playstation_breach_costs/

[26] U.S. Department of Health & Human Services. (2012, June) Alaska settles HIPAA security case for $1,700,000. [Online]. http://www.hhs.gov/news/press/2012pres/06/20120626a.html

[27] Évariste Galois, "Œuvres Mathématiques," *Journal de Liouville*, 1846. [Online]. http://perso.univ-rennes1.fr/antoine.chambert-loir/DJVU/

[28] Évariste Galois, "Œuvres Mathématiques," *Société Mathématique de France*, 1897. [Online]. https://ia600300.us.archive.org/3/items/uvresmathmatiqu00frangoog/uvresmathmatiqu00frangoog.pdf

[29] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. (2001, August) Handbook of Applied Cryptography. [Online]. http://cacr.uwaterloo.ca/hac/

[30] Nik Cubrilovic. (2009, July) The Anatomy Of The Twitter Attack. [Online]. http://techcrunch.com/2009/07/19/the-anatomy-of-the-twitter-attack

[31] Lorenzo Martignoni et al., "Cloud Terminal: Secure Access to Sensitive Applications from Untrusted Systems," in *USENIX ATC'12 Proceedings of the 2012 USENIX conference on Annual Technical Conference*, Berkeley, CA, 2012, p. 12. [Online]. http://dl.acm.org/citation.cfm?id=2342835

[32] Jeff Barr, Attila Narin, and and Jinesh Varia, "Building Fault-Tolerant Applications on AWS ," Amazon Web Services, Whitepaper 2011. [Online]. http://media.amazonwebservices.com/AWS_Building_Fault_Tolerant_Applications.pdf

[33] Legislative Assembly of the Province of British Columbia, "Bill 73 - Freedom of Information and Protection of Privacy Amendment Act, 2004," Law 2004. [Online]. http://www.leg.bc.ca/37th5th/3rd_read/gov73-3.htm

[34] University of Pennsylvania - Information Systems & Computing. (2013, June) Computer Power Usage. [Online]. https://secure.www.upenn.edu/computing/resources/category/hardware/article/computer-power-usage

[35] Rogers Communications. (2014, January) Internet Package Options. [Online]. http://www.rogers.com/web/link/hispeedBrowseFlowDefaultPlans

[36] Bell Canada. (2014, January) Bell Internet packages. [Online]. http://www.bell.ca/Bell_Internet/Internet_access

[37] TekSavvy Solutions Inc. (2014, January) Cable Internet Packages. [Online]. http://teksavvy.com/en/residential/internet/cable

[38] Shaw Communications. (2014, January) Shaw Internet. [Online]. https://www.shaw.ca/internet/plans/

[39] AT&T. (2014, January) U-verse High Speed Internet. [Online]. http://www.att.com/shop/internet/u-verse-internet.html

[40] Verizon. (2014, January) FiOS Internet - Dramatic speed for lots of action. [Online]. http://www.verizon.com/home/fios-fastest-internet/

[41] Comcast. (2014, January) XFINITY Internet. [Online]. http://www.comcast.com/internet-service.html

[42] Dropbox. (2014) Upgrade to Dropbox Pro. [Online]. https://www.dropbox.com/upgrade

[43] Amazon Web Services Inc. (2014) Amazon S3 Pricing. [Online]. http://aws.amazon.com/s3/pricing

[44] Microsoft. (2014) Compare OneDrive Pricing. [Online]. https://onedrive.live.com/about/en-us/compare

[45] Apple Inc. (2014) iCloud storage plan overview. [Online]. http://support.apple.com/kb/PH12796

[46] Google. (2014) Storage plan pricing. [Online]. https://support.google.com/drive/answer/2375123

[47] PCPartPicker. (2014, January) PCPartPicker - Storage. [Online]. http://ca.pcpartpicker.com/parts/internal-hard-drive/

[48] Richard W. Hamming, "Error Detecting and Error Correcting Codes," *The Bell System Technical Journal*, vol. 29, no. 2, pp. 147-160, April 1950. [Online]. http://onlinelibrary.wiley.com/doi/10.1002/j.1538-7305.1950.tb00463.x/abstract

[49] Edward K. Lee Peter M. Chen, Garth A. Gibson, Randy H. Katz, and David A. Patterson, "RAID: High-Performance, Reliable Secondary Storage," *ACM Computing Surveys*, vol. 26, pp. 145-185, 1994. [Online]. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.41.3889&rep=rep1&type=pdf

[50] Robert G. Gallager, "Low-Density Parity-Check Codes," *Information Theory, IRE Transactions on*, vol. 8, no. 1, pp. 21-28, January 1963. [Online]. http://www.rle.mit.edu/rgallager/documents/ldpc.pdf

[51] Adi Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612-613, 1979. [Online]. http://dl.acm.org/citation.cfm?id=359176

[52] Michael O. Rabin, "Efficient Dispersal of Information for Security, Load Balancing, and Fault Tolerance," *Journal of the ACM (JACM)*, vol. 36, no. 2, pp. 335-348, 1989. [Online]. http://dl.acm.org/citation.cfm?id=62050

[53] Irving S. Reed and Gustave Solomon, "Polynomial Codes Over Certain Finite Fields," *Journal of the Society for Industrial and Applied Mathematics*, vol. 8, no. 2, pp. 300-304, June 1960. [Online]. http://epubs.siam.org/doi/abs/10.1137/0108018

[54] CareCloud Corporation. (2014) HIPAA Compliant Cloud Storage. [Online]. http://www.carecloud.com/hipaa-compliant-cloud-storage/

[55] US Department of Health and Humane Services. (2007, March) HIPAA Security Standards: Physical Safeguards. [Online]. http://www.hhs.gov/ocr/privacy/hipaa/administrative/securityrule/physsafeguards.pdf

[56] US Department of Health & Human Services. (2007, March) HIPAA Security Standards: Technical Safeguards. [Online]. http://www.hhs.gov/ocr/privacy/hipaa/administrative/securityrule/techsafeguards.pdf

[57] Sudhir K Bansal. (2014, January) How to encrypt your files before uploading to Cloud Storage using CloudFogger. [Online]. http://thehackernews.com/2014/01/how-to-encrypt-your-files-before.html

[58] Ned Dimitrov, "CS337 Project 1: Information Dispersal," University of Texas at Austin, Austin, TX, USA, 2004. [Online]. http://www.cs.utexas.edu/~plaxton/c/337/05s/projects/1/desc.pdf

[59] Todd Mateer, "Efficient algorithms for decoding Reed-Solomon codes with erasures," Clemson University, Clemson, SC, 2014. [Online]. https://mthsc.clemson.edu/misc/MAM_2014/bmj9b.pdf

[60] Hugo Krawczyk, "Secret Sharing Made Short," in *CRYPTO '93 Proceedings of the 13th Annual International Cryptology Conference on Advances in Cryptology*, London, UK, 1993, pp. 136-146. [Online]. http://www.cs.cornell.edu/courses/cs754/2001fa/secretshort.pdf

[61] Athicha Muthitacharoen, Robert Morris, Thomer M. Gil, and Benjie Chen, "Ivy: A Read/Write Peer-to-Peer File System," *ACM SIGOPS Operating Systems Review - OSDI '02: Proceedings of the 5th symposium on Operating systems design and implementation*, vol. 36, no. SI, pp. 31-44, December 2002. [Online]. http://pdos.csail.mit.edu/ivy/osdi02.pdf

[62] Certes Networks. (2011) Transparent, High-Speed Data Center Security. [Online]. http://www.certesnetworks.com/pdf/1G-10G-DataCenter-SN.pdf

[63] Sun Microsystems Inc. (1989, March) NFS: Network File System Protocol Specification. [Online]. http://tools.ietf.org/pdf/rfc1094.pdf

[64] Red Hat Inc. (2009, November) GlusterFS - About. [Online]. http://www.gluster.org/about

[65] Ion Stoica, Robert Morris, David Karger, Frans Kaashoek, and Hari Balakrishnan, "Chord: A Scalable Peer-to-peer Lookup Service for Internet," in *SIGCOMM '01, 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, San Diego, CA, August 2001, pp. 149-160. [Online]. http://pdos.csail.mit.edu/papers/chord:sigcomm01/chord_sigcomm.pdf

[66] James S. Plank. (2004, August) All About Erasure Codes: Reed-Solomon Coding, LDPC Coding. [Online]. http://web.eecs.utk.edu/~mbeck/classes/cs560/560/notes/Erasure/2004-ICL.pdf