Robust and Traffic Analysis Resistant Cloud File System

Madhvi Gupta

MT10009

IIIT-D-MTech-CS-IS-11-001

Oct 12, 2012

Indraprastha Institute of Information Technology
New Delhi

Thesis Committee
Shishir Nagaraja (Chair)
Somitra Sanadhya
Prof. Balakrishnan

Submitted in partial fulfillment of the requirements
for the Degree of M.Tech. in Computer Science,
with specialization in Information Security

Keywords: Cloud computing, Encryption, Erasure codes, Steganographic file system, System administrator.

Certificate

This is to certify that the thesis titled **"Robust and Traffic Analysis Resistant Cloud File System"** submitted by **Madhvi Gupta** for the partial fulfillment of the requirements for the degree of *Master of Technology* in *Computer Science & Engineering* is a record of the bonafide work carried out by her under my guidance and supervision in the Security and Privacy group at Indraprastha Institute of Information Technology, Delhi. This work has not been submitted anywhere else for the reward of any other degree.

Advisor Name
Dr. Shishir Nagaraja

Indraprastha Institute of Information Technology, New Delhi

Abstract

With improved technology, every user now generates huge amount of data that requires ever increasing amount of space to store it. It is not economical for the users to purchase new storage device every time. As a remedy to this problem different service provider offers storage space that can be utilized by the users. Cloud computing also provides a similar functionality to the users for storing their data over the Internet. The invention of cloud computing has fulfilled the need for extra storage space but at the same time it has also lead to increased concerns regarding security of data stored over the Internet.

The responsibility of the security of this huge amount of data lies with the service provider, but there may arise a situation where user may not trust the service provider due to the sensitivity of data. So there is a need of a technique which provide users a level of trust and security.

In Document security solution (DSSol), we have tried to deal with such a situation, where we provide protection against adversary which can also be the service provider itself. We have used the Steganographic file system along with encryption to protect the document uploaded over the Google Docs from any security breach. The design encrypts the data files to be uploaded and divides them in equal sized chunks which gets uploaded along with some additional dummy chunks to the Goggle Docs. This provides unobservability, traffic analysis resistance and thereby providing protection against attacks possible due to traffic analysis and usage patterns.

Acknowledgments

Any accomplishment requires the effort of many people and this thesis is no different. It gives me immense pleasure in recording my appreciation and a deep sense of gratitude to all the individuals who extended their unstrained cooperation in completing my thesis.

I express my sincere thanks and gratitude towards Dr. Shishir Nagaraja (Advisor) for his cooperation and guidance extended to me and making this thesis possible.

With this, I also feel highly obliged to all fellow colleagues who were always there to provide me direction and were solving my problems and confusions in a way to enable me to develop the thesis in an efficient way.

Madhvi Gupta

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Security of user information has always been a major concern especially when storing and retrieving data over the Internet. With the introduction of cloud storage services this concern has grown even deeper as the data stored over the Internet has multiplied many folds. There has been several instances where the user suffered from either data loss or the security of their data got compromised. These issues are also inherited in the emerging technology of cloud computing. In April 2011, Amazon Web Services (AWS), Elastic Block Store (EBS), which stores Amazon Elastic Compute Cloud (EC2) instances, experienced an outage that lasted for four days. The outage lead to a loss of 0.07 percent of the total data in US-East Region which Amazon stated was not fully recoverable [1]. In another instance Bloomberg News reported that hackers used AWS's EC2 cloud computing unit to launch an attack against Sony's PlayStation Network and Qriocity entertainment networks. The attack reportedly compromised the personal accounts of more than 100 million Sony customers [2].

1.1 Cloud

Cloud is a large pool of easily usable and accessible virtualized resources (hardware, development platform and services). These resources can be dynamically reconfigured to adjust to a variable load (scale), allowing for an optimal resource utilization. This pool of resources is typically exploited by a pay-per-use model and is also known as "On demand computing".

Cloud computing has emerged as a perfect solution for the growing storage demand and has gained popularity among the users which is also evident from a survey International Data Corporation (IDC) conducted in 2010. It was observed that the cloud has become the first choice for the users Fig 1.1 [4]. Service providers also sensed the current need and started investing in cloud computing Table 1.2 [4]. Some of the cloud service providers include organizations like AMD, Cisco, Citrix, EMC, HP, IBM, Intel, Microsoft, Novell, Rackspace, RedHat, Savvis, Sun Microsystems and VMware, etc.

Cloud computing encompasses a subscription-based service that in real time over the Internet, extends IT's existing capabilities. A simple example of cloud service is Yahoo Mail, GMail

Figure 1.1: Cloud Security Survey by IDC.

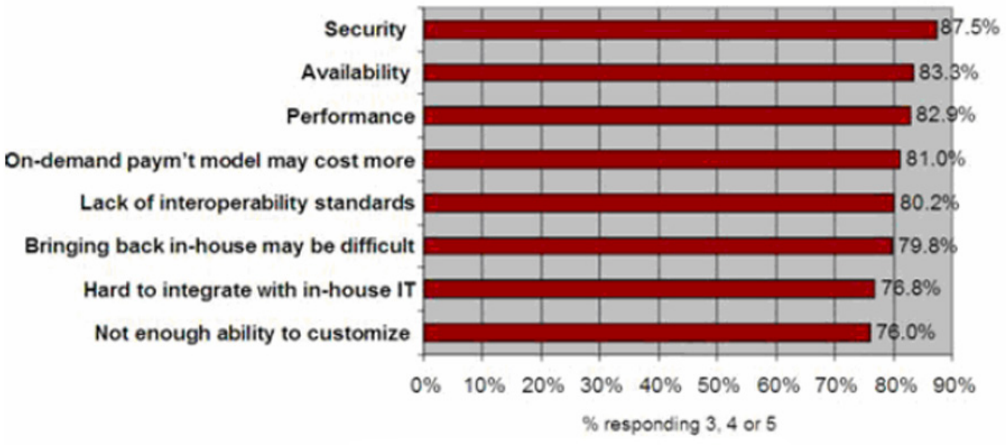


Figure 1.2: Cloud Growth According to survey by IDC.

Year	2008	2012	Growth
Cloud IT Spending	$ 16 B	$42 B	27%
Total IT spending	$383 B	$ 494 B	7%
Total-cloud spend	$367 B	$ 452 B	4%
Cloud Total spend	4%	9%	

etc. All a user need is just an Internet connection and he can start sending emails. The server and email management software is all in the cloud (Internet) and is totally managed by the cloud service provider. Cloud computing inherits a number of advantages from its predecessor computing technologies like distributed and grid computing. Some of these major advantages are:

1. Resilience: Cloud is distributive in nature and hence removes single point of failure. The failure of one node does not affect the information or the service availability, therefore providing a highly resilient computing environment [3].

2. Scalability: With cloud computing, user can lease infrastructure resources on-demand from a virtually unlimited pool. User pays only for what he is actually using but not that is available. This way, a user can optimize his IT investment and improve availability and scalability [3].

3. Flexibility and efficiency: Cloud computing facilitates expanding or contracting computing power as per the demand. It allows effective load balancing and flexibility which ensures that the performance of processes and transactions remain as if the user has resource at his own local machine [3].

1.1.1 Background of cloud

The term Cloud was first used in the telephone industry in the early 1990s when the Virtual Private Network (VPN) service was being offered for the first time which replaced the hardware data circuits between the service provider and customers for data communication [5]. VPN provided the service providers with an efficient tool to utilize the available amount of bandwidth to fulfill their consumer needs at a lower cost. The network traffic could now be rerouted in real time to maximize the network utilization making the data path unpredictable. As a result, the service providers represented their network responsibilities by a cloud symbol. The same infrastructure concept later developed as cloud computing. Formally cloud computing as defined by NIST [6] states cloud computing model is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g. networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.

Different types of service models can be defined for cloud computing [7] which are:

1. Private Cloud Service Models: The private cloud is owned by a particular organization or a group of organizations and provides a virtualized infrastructure, which can be shared by different departments, hence minimizes the need for separate resources for each department. Such private clouds can also be further extended to partners, contractors, suppliers, etc. The private cloud is more secure as it is only exposed to a limited number of users.

2. Public Cloud Service Models: The public cloud on contrary to the private cloud is not owned by any single organization or a group. It operates through the different peer-to-peer IP based networks. It is a multi tenant environment offered by a service provider and have a greater exposure to security risks than a private cloud.

3. Hybrid Cloud Service Models: Hybrid clouds are a combination of public and private clouds. In such clouds, highly critical and confidential data are maintained in the private cloud, while less critical data is stored in the public cloud. Therefore the hybrid cloud has both the features of the public and private cloud with a choice to choose between the two based on different resource, control, and security needs.

Three services delivery models are provided by the cloud computing [5]

1. Infrastructure-as-a-service (IaaS): In IaaS, service provider offers a Virtual machine (VM) platform and different computation resources like CPU, memory, storage, and networking, as an Internet based services. Amazon EC2 is the most popular IaaS provider.

2. Platform-as-a-service (PaaS): In PaaS the consumer utilizes different platforms, tools and other business services provided by the service provider to develop, deploy, and manage his applications without installing any of these platforms or support tools on his local machine. Google Apps and Microsoft Windows Azure are most known for this service.

3. Software-as-a-service (SaaS): In SaaS the customer uses the software applications hosted by the service provider of the cloud infrastructure. SalesForce CRM is one of such service providers.

1.2 Problem Statement

A cloud service provider though claims to offer a simple, clean and secure file system interface for storing data on their servers but the service provider may itself suffer with problems of unavailability and security. The service providers don't have any transparency in their criteria of optimization of space. Even it is possible that if data is moved to a certain location then residual data might be left with the threat of unauthorized access.

In the present scheme we limit our scope to secure storage and retrieval of user information over Google Docs. In June 2007, Google was ranked as the worst offender according Privacy International, a U.K. based privacy rights watchdog, which ranked the 23 privacy offender companies as "Don't Be Evil" [8] .

Google dominates the online search engine market with an estimated share of about 70% in the online search engine market. With such a huge share and its access to user information , the concern for secure storage of user information in Google environment grows even deeper.

As per the Google terms of service (March 1, 2012) Privacy and Copyright Protection

"By using our Services, you agree that Google can use such data in accordance with our privacy policies."

Your Content in our Services

When you upload or otherwise submit content to our Services, you give Google (and those we work with) a worldwide license to use, host, store, reproduce, modify, create derivative works (such as those resulting from translations, adaptations or other changes we make so that your content works better with our Services), communicate, publish, publicly perform, publicly display and distribute such content. The rights you grant in this license are for the limited purpose of operating, promoting, and improving our Services, and to develop new ones. This license continues even if you stop using our Services (for example, for a business listing you have added to Google Maps)." `http://www.google.com/intl/en/policies/terms/`

Adding to this, Google may also carry out surveillance to get information about the important files for a particular user and how many times users have accessed those files. For example: Goggle Timestamp: Every time we upload or down a file on Goggle Docs a timestamp gets associated with the data which reveals information about it.

So, there is a need for alternative which not only allows the users to avail these services but at the same time also ensures the security of their data. **The main idea of the present design is to provide protection and unobservability for user's data files against an adversary (Google) who can learn the traffic patterns and frequency of file access (number**

of times file is accessed). It aims to develop a system for protecting the user file stored on Google Docs from attacks possible due to monitoring and traceability based on an analysis of usage patterns. The design follows a proactive approach which prevents the exposure of any kind of confidential information or the identity of important files from not only an adversary but from the system administrator itself. Another distinguishing feature which the scheme provides is to hide the traffic patterns of the usage of the files.

Chapter 2

Literature Review

2.1 Steganographic file systems

The concept of a steganographic file system was first proposed in the paper "The steganographic file system" [10] by Anderson et al. The objective of this design was to protect against a situation where a user is compelled to reveal the keys or the contents of his file. The key idea behind this approach was that, rather than hiding the data in image or audio files the data are hidden on the disk directly. The file system is kept on the disk itself and an object name and password is attached to it for its security. The user with the appropriate authentication, that is the correct object name and password will only be allowed to access the file. On the other hand an adversary who does not have the authenticated object name and password cannot even deduce that whether a file with such object name exists or not. Empty space on the storage device cannot be claimed but it is possible to reveal some less sensitive files, as adversary cannot determine how much data have been hidden on the storage device. Two approaches were proposed for the building of such system [10]. In the first approach information is too hidden in cover files that can be retrieved by XORing a subset of the files. In the second approach, complete hard disk is filled with random bits and the address calculation for storing that file block is done by using a pseudorandom process like encrypting the block's location in the file using a key, a one way hash of the filename and password etc. The second approach described in [10] was carried forward by Van Schaik and Smeddle [11] and resulted in the implementation of a file system. But they were not able to achieve security and plausibile deniability which were their design goals. They neither hide the information by linearly combining password selected subsets of blocks nor they replicated them. In continuation to this a similar approach for linux system was proposed by Kuhn and McDonald in their paper "StegFS: A Steganographic file system for Linux" [12]. They emphasize on the use of block allocation table (BAT). The proposed scheme provides security at different levels i.e. if one file is decrypted, all the files at lower levels in the BAT are also decrypted. The BAT table has two entries. The first one contains the file and another has the key. Each file is encrypted using its corresponding key entry in the table. After that Zhou, Pang and Tan proposed new approach for hiding the files on the disks in [13]. In this approach

blocks are scanned to check whether they were allocated or free. The header generated for every hidden file is saved in a pseudo random free location. The location for storing the file is derived from its name and a secret key. For retrieving the file, the header is used as it contains the links to the locations where file blocks resides. The scope of all the above approaches was limited to hiding the files on the local machine only. So, Hand and Roscoe proposed Mnemosyne [14] to extend the concept of the Steganographic file system to distributed systems. In this approach the location for storing a file is chosen randomly by hashing the file name, block number and a secret key. Another approach in this domain was contributed by Giefer and Letchner named as Mojitos [15] which combined the ideas of Mnemosyne (by Hand and Roscoe [14]) and Steg SFS (by Kuhn and McDonald [12]). According to Giefer and Letchner [15] most of the previous SFS proposals were designed to protect against attackers who obtained the snapshots of the file stored sufficiently spaced in time. However, there are scenarios where adversaries such as system administrator permanently monitor the file stored and use the traffic information to obtain evidence about the hidden files. StegFS [12] and "A Framework for the Analysis of Mix Based Steganographic File Systems" [9] by Claudia et al. are the only previous proposal of SFS that implement measures to protect against this adversary model. The latter develops a framework for analyzing mix based SFSs. It uses a local mix to relocate files in the remote store, and thus protect from the known attacks against low entropy block techniques. The context of such file system is also extended to Google docs like GSpace, gVault, etc.

gVault provides such an alternative to secure and preserve user data while using such services. gVault [16] is a cryptographic network file system which utilizes the data storage provided by Gmails' web based email service. It proposes a design of a storage model which does not reveal any information about content, metadata and structure of the file system. gVault runs an application on the client's local machine only and trusts the local machine as any other thing can be malicious. The application requires the user to enter his Gmail username and Gmail password. In addition, it also asks for the master password that adds another level of security. Once this authentication is done gVault opens a session with the Gmail server. The user can then perform all the operations that he wants to perform on the client side which are then transformed into their equivalent HTTP request. The responses received from the server are then parsed by the application to get the user's data which is encapsulated in the HTTP request. The scheme also provides operations like creating, updating, moving, deleting a file or directory , etc. Also, the application provides a user friendly GUI and only runs on client side with no modification required at server side.

Gspace turns the 7GB storage space of any Gmail account into free online storage which can be used to manage unlimited Gmail accounts to store all type of files within its simple, user friendly interface. Gmail Filesystem provides a mountable Linux filesystem which uses the user's Gmail account as its storage medium. Gmail Filesystem is a Python application that uses the FUSE userland filesystem infrastructure to provide a Linux filesystem, and libgmail to communicate with Gmail.

Claudia, Carmela and Bart studied the security of Steganographic File systems (SFSs) against

an adversary who monitors the accesses to the files store [9]. The architecture presented makes use of pool mixes of mixing the legitimate file blocks with the dummy blocks. The purpose behind the pool mixes is to achieve high entropy block relocation so as to prevent from the traffic analysis attack [18] which uses low entropy block relocation algorithms [19]. Also they have introduced probabilistic metrics to quantify unobservability and deniability provided by SFSs against coercion attacks.

2.2 Cloud Storage Security Assurance

In cloud storage environment, the client wants to get rid of the data storage and hence stores his data on unreliable servers but at the same time he also wants to ensure that the data has not been tampered or deleted by the cloud service provider without downloading the entire data as it is quite inefficient. Ateniese et al. [20] have formalized a model called provable data possession (PDP), for solving this issue. The file is preprocessed by the client and the metadata that will be used for verification is produced. The file is then sent to the non reliable server for storage purpose. The client can then send some random challenge to the server for the verification of authenticity of data. The server can then prove that the data has not been tampered by providing the appropriate reply to the challenge.

Juels and Kaliski presented the proofs of reliability (PORs) [17]. This proof can be considered as cryptographic proof of knowledge primarily designed for large files. In this scheme the file is first preprocessed and then encrypted. After this the file is sent for storage on the server. Special sentinel blocks are randomly inserted to detect corruption of data and error correcting codes are used to recover from the data corruption which improves the resiliency of the system. But this scheme was unable to support updates in the file, instead it simply replaces the updated version of the file with existing versions of the file. In addition there exists a prior upper bound on the number of queries which can be performed by the client. The PDP schemes [20–23] are applicable to static files only. In this no update or modification can be done in a file once it is stored on the server. Dynamic Provable Data Possession [25] tried to provide support for dynamic operations on files that is a dynamic provable data possession which allows provable updates on the stored data. The scheme proposed an update function on a file F with n blocks, that can be used for either insertion of a new block or modification of an existing block, or deletion of any block. For the verification of the integrity of the file blocks in the implementation of DPDP (dynamic provable data possession) authors used the modification of authenticated skip list data structure [24] which is called as rank based authentication skip list. Also, Ateniese et al. presented a dynamic PDP solution knows as Scalable PDP [27]. The design aims to develop a light weight and provably secure PDP scheme and come up with all future challenges during setup and store precomputed answers as metadata. The scheme is dependent on symmetric key cryptography. It also supports update function which makes it suitable for realistic applications. The limitation with this solution is an upper bound on the number of challenges that can be sent by the client and the maximum number of updates that can be

performed by him. Further, block insertion is possible in an append type mode. Each update requires recreating of the remaining challenges which make it unsuitable for large files. Kevin, Jules and Alina proposed HAIL (A High Availability and integrity Layer for Cloud Storage) [28]. It is designed like a proactive cryptographic system against a mobile adversary which is capable of corrupting all the service providers but not simultaneous and can progressively attack the storage devices. HAIL tries to manage the integrity and availability across a collection of servers, as in a distributed environment a file is spread across the different servers with redundancy to ensure retrievability if one or more servers fails. HAIL make use of PORs to test the storage resources and in case of any failures reallocation is supported. It extends the single server design of PORs to both within server redundancy and cross server redundancy. It makes use of a client side application for the purpose of verification. The objective of HAIL is to protect integrity rather than secrecy, so the design strategy of Test and Redistribute (TAR). With this model, the client uses POR to identify the corruption of file blocks by sending a challenge to the server and when required reallocate the resources. If a fault occurs on a server then client communicates with other servers in order to recover the corrupted blocks of the file and resets the faulty server with correct share.

Another set of researcher contributed towards revealing the vulnerabilities when cloud infrastructure shares its physical resources among different users by using virtual machines [29]. In a cloud computing environment, multiple VMs are used for maximizing the efficiency and providing computing and software capabilities requested by the user using a single physical machine. The assigning of multiple VMs to same physical machine on the server introduces a threat, that is if any adversary VM also get assigned to the same physical server machine as one of the customers. The adversary requires a two step attack to breach the confidentiality of the customer. First step is the placement in which the adversary tries to place their malicious VM on the same physical machine as their target user. Once the adversary successfully completes the first step, the second step is the extraction which refers to the extraction of the confidential information through a cross-VM attack. This whole study was concentrated on Amazon EC2 cloud. Firstly they showed that it is possible to identify where an instance is located in the cloud infrastructure. The instance location can be traced as different availability zones correspond to different internal IP address range which are same as the ID used by the user for the identity of his instance. After identifying the location where the instance is stored the next step is to determine whether the two instances are co-resident on the same physical machine or not. Co-resident instances are determined by a number of parameters like if they are matching IP address, calculating the round trip time for small packets or numerically close internal IP address. Now a successful attack will take place if an adversary will be able to launch instances that will be the co-resident to the instance of the target user. This observation leads to the fact that a single account will never have two instances simultaneously on the same physical machine. So, if a user has n instances under a single account they will be placed on n separate machines. The observation also showed the strong placement locality that is if two instances are running sequentially then they will most probably reside on the same machine. The last step will be to

utilize the side channels to extract information about co-resident instances. Also, these channels can be used to measure the cache usage and traffic rate which in turn are used for different kind of attacks like keystroke timing attacks. The best solution suggested is to reveal this risk to the customers so that depending on the criticality of the information stored or accessed, the cloud users can take proactive actions like using physical machines for their own VMs, etc.

2.3 Erasure Codes

The need of computational power, storage and communication bandwidth gave rise to the development of distributed system in place of the centralized system. The distributed system consists of a number of cooperative computer systems which collectively provides the quantity and quality of service promised to the users. All these distributed components in such systems cannot be available all the time to serve the users because of one reason or the other. Hence, sometimes there arises a situation of reduced availability where one or more system is not available at the time when the user request for a particular service e.g. user wants to retrieve his data. The concept of replication arises as a remedy to this situation. But such replication is also not much effective and results in wastage of resources. More effective solution for this problem is provided by erasure codes [30], which provides redundancy with lower overhead and makes the system, failure resistance. Erasure codes divide a block of data into a number of fragments say m, which are then used to produce n number of redundant fragments, such that n $>$ m. For the reconstruction of any block of data only m number of fragments are needed out of the total n number of fragments. This scheme is known as an erasure coding scheme of m of n. A number of variants for the erasure codes have been suggested over a period of time like Reed Solomon [31], tornado erasure codes [32], Weaver [26].

Reed Solomon codes are error correction codes invented in 1950 by Irving S. Reed and Gustave Solomon at MIT laboratory [31] and for retrieving its original data even if some amount of data is missing. The encoding and decoding is carried out in RS codes by performing arithmetic operation over Galois fields.

The problem definition for RS codes can be summarized as follows:

Let n storage devices hold data: D_1, . . . , D_n each with size k bytes. m storage devices hold checksums:C_1 , . . . , C_m each with size k bytes. The checksums are computed from the data using a function F Fig 2.1.

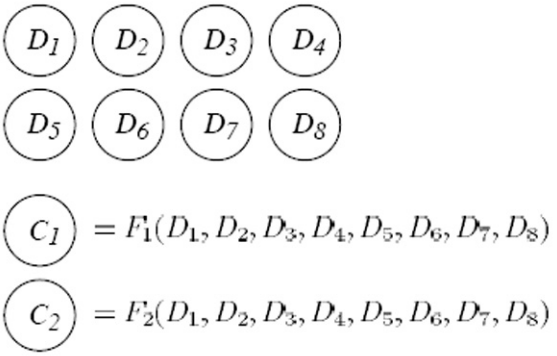
Goal: if any m devices fail, they can be reconstructed from the surviving devices.

The algorithm works as follows:

1. Division of data into strips of equal size: The encoding in RS scheme breaks the data into words of size w bits and works on stripes whose width is w as shown in Fig 2.2.

2. Checksum calculation: Considering a single strip, the data words are d_1, . . . , d_n and checksum words are c_1 , . . . , c_m.

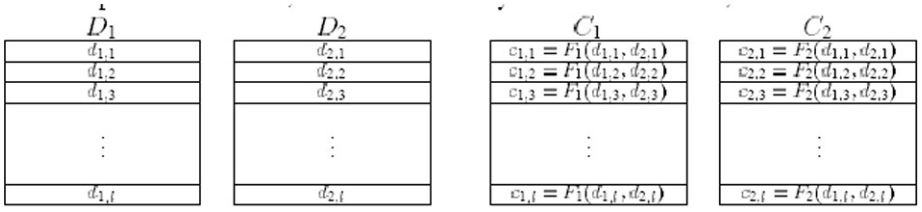
Figure 2.1: Providing two site fault tolerance with two checksum devices.



$$\begin{bmatrix} f_{1,1} & f_{1,2} & \dots & f_{1,n} \\ f_{2,1} & f_{2,2} & \dots & f_{2,n} \\ \vdots & \vdots & & \vdots \\ f_{m,1} & f_{m,2} & \dots & f_{m,n} \end{bmatrix} \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_n \end{bmatrix} =$$

$$\begin{bmatrix} 1 & 1 & 1 & \dots & 1 \\ 1 & 2 & 3 & \dots & n \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & 2^{m-1} & 3^{m-1} & \dots & n^{m-1} \end{bmatrix} \begin{bmatrix} d_1 \\ d_2 \\ \vdots \\ d_n \end{bmatrix} = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_m \end{bmatrix}.$$

Figure 2.2: Breaking storage devices into words.



Now FD=C

where F has defined a linear function.

D is a data strip of w bytes

C is checksum of w bytes.

The equation can be expanded as c_i = F(d_1,d_2,...,d_n).

If any of the data bits while transforming is changed to d_j‘ then computation of d_j is carried out according to the equation

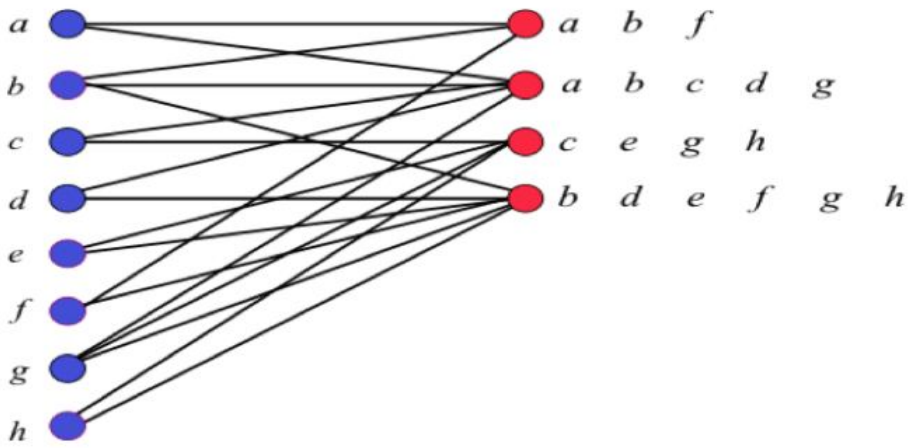
c_i‘ = $G_{i,j}(d_j, d_j$‘, $c_i)$=c_i + $f_{i,j}(d_j$‘ − $d_j)$

3. Data retrieval: While retrieving the data even if m rows are removed we can recover the data as per the below matrix.

Tornado codes are based on irregular sparse graphs which are generated by cascading a sequence of irregular random bipartite graphs connecting data nodes to check nodes.

Encoding: Process of encoding is based upon LDPC(low density parity check) The LDPC (low density parity check) coding scheme is used at each level of an erasure code with the left nodes being used to calculate the right parity check nodes. The LDPC scheme was originally described by Luby [26]. The basic unit of a Tornado Code is a bipartite low-density parity check graph. A number of data nodes represent the data to be stored. These data nodes are then used to calculate the check nodes to provide redundancy. The check nodes are calculated by XORing the data nodes Fig 2.3.

Figure 2.3: Encoding Process



Decoding: The process of the reconstruction of the data is just the reverse of construction of the graphs. If any right node has exactly one missing left node, the missing left node can be reconstructed. The parameters involved in the construction of a Tornado Code include the number of nodes, the number of levels, and the left and right edge distributions between each level.

Chapter 3

Threat Model

The storage need of user now-a-days has increased many folds due to the proliferation of digital devices in his life. The solution to this storage problem is provided by a number of organizations like Google, Amazon , etc., by providing cloud storage. Google provides a huge amount of storage at no cost, which makes it lucrative to most of the users. The requirement here is not just to store the data but to store it securely. Cloud is a third party service thereby giving rise to the question of trust. The extent a client can trust the cloud service provider for storing his data depends on the sensitivity of the data.

CERT provides some of the attacks that a malicious system administrator may perform [33].

1. Threat to data security: The system administrator at a data-mining firm had access to servers and data owned by the victim organization to process customer information. An unprotected file containing encrypted password information was found on one of these servers. The administrator, brute-force attacked over 300 passwords, accessed data belonging to dozens of the victim organization‘s customers, and downloaded millions of personal records. Fortunately, the information was never sold or released by the insider prior to arrest.

2. Threat to data availability: Another often-overlooked threat posed by the malicious administrator is impact of customer data availability. An incident captured by CERT described a system administrator who rather than compromises customer information, simply removed critical software from the hosting systems, which prevented the provider from responding to customer requests. While customer data remained intact and confidential, customers were adversely affected by their lack of access to important information.

Data theft attacks are considered to be one among the top threats to the cloud computing by Cloud Security Alliance [34]. So, if the administrator is malicious the harm can be significant to security of data. He has the capability to learn traffic patterns of a particular user, no of accesses user has made to which file and also can identify the file sizes. Another incident of data theft took place on Twitter where corporate and personal documents were exfiltrated to technological

website TechCrunch [35] and customer‘s accounts were illegally accessed. The attacker used a Twitter administrator‘s password to gain access to Twitter corporate documents, hosted on Google infrastructure as Google Docs. The damage was significant both for Twitter and for its customers.

Above attack scenarios lead to the fact that storing unencrypted data in the cloud is not secure. If the system administrator is malicious then he can access the user data and hence compromises the data confidentiality. Also, a malicious cloud provider (system administrator) can modify or alter the user‘s data that will lead to a loss of integrity of data. These arguments lead to the necessity of encryption.

Breaking the security of any file system doesn‘t necessarily need decryption of the data, it only requires proving that the important data exists. Hence encryption alone cannot prevent the leakage of private information. Even if the files are encrypted a correlation between the unencrypted file size and encrypted file size can easily be established which enhances the adversary capability to mount an attack for decrypting the important file. Hence encryption alone won‘t solve the purpose of providing security to the file system.

Applications generally follow the CIA (Confidentiality, Integrity, Authenticity) model but it does not prevent leakage of private information. For example:

1. File size: The file size can be used as a distinctive feature for identifying a particular file.
2. No. of accesses: It can alone be used as a parameter to determine the set of important files from a whole set of files.
3. Traffic patterns: It reveals the usage patterns of the user that can be combined with other parameters to mount an attack.

The proposed solution protects the user from an adversary such as a malicious system administrator. The objectives of the proposed design are:

1. Unobservability: Given a file system in Google Docs, the unobservability is defined as the difficultly to identify if there is a file stored or not on the Google Docs.
2. Robustness: Design should be robust against the deletion of files from Google Docs.
3. Traffic analysis resistant: The adversary can learn traffic patterns of the user. The design should provide a mechanism to make it resistant to traffic analysis.

Chapter 4

Design

Our design aims to follow a proactive approach which prevents the exposure of any kind of confidential information or the identity of our important files from not only any adversary but also from the system administrator.

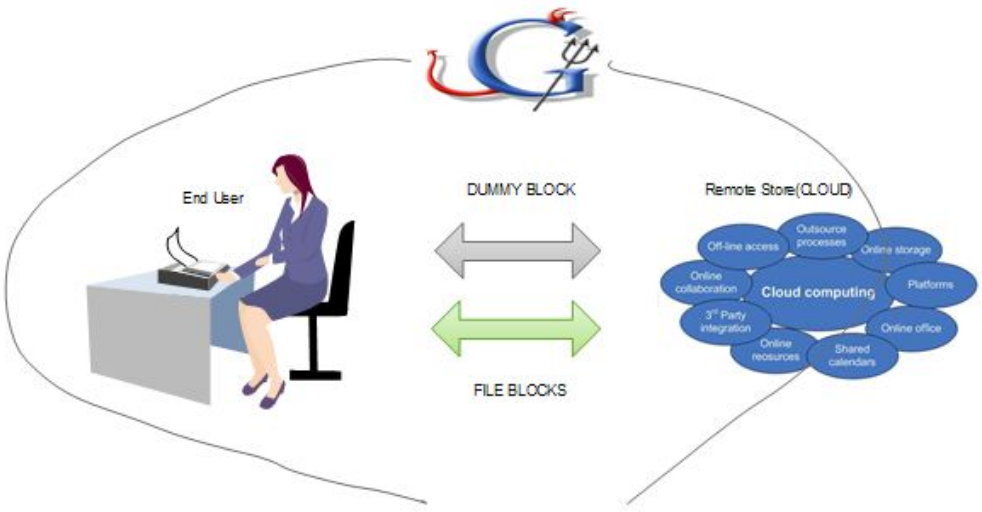
Main terms used in the design are:

1. Chunk: A chunk is any file of a defined size. A chunk can be a derived from an original file or can be a dummy chunk.
2. Chunk Size: It is the size defined for any chunk. In this scheme a default size of 256 KB is taken for every chunk. File with greater size than the default chunk size are divided and padded (if required) with additional bits to convert them into equal chunks of 256 KB each.
3. Legitimate chunks: The file is being divided into small chunks (256 KB each) which are referred as the legitimate chunks. These chunks are uploaded to the Google Docs.
4. Dummy chunks: Dummy chunks contain randomly generated garbage data with a chunk size of 256 KB each. These are uploaded and downloaded along with the legitimate chunks.
5. Metadata files: Metadata is the data about files. In the present scheme the metadata files store the information (explained in 4.2) required for the processing of different tasks carried out in the design.

The architecture consists of following components Fig 4.1:

1. End User: He is the legitimate user who has account to use Google docs.
2. Secure Application: The application is responsible for encryption of file, splitting file into chunks, creating dummy chunks, storing it on Google cloud, fetching blocks of a file and combining the chunks.

Figure 4.1: Architecture



3. Local Disk: This is a local trusted machine of the user where the application is deployed. The files required to be stored on the Google Docs are available on this machine.

4. Channel: The network through which the file is sent to Google Docs.

5. Google Docs: Google Docs is the service provided by Google to store the files.

4.1 Approach

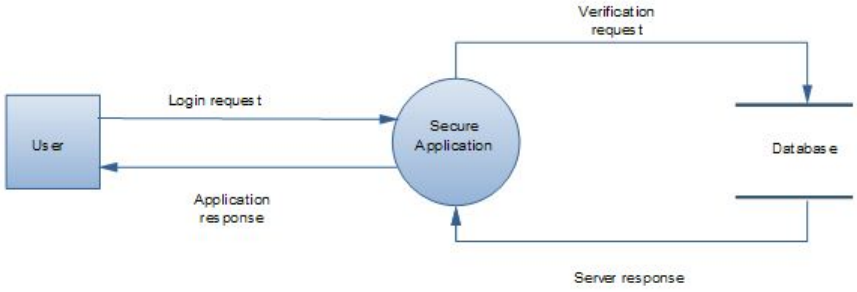
The proposed design is based on providing unobservability for the user‘s data files. It introduces equally sized dummy chunks along with legitimate chunks, which makes it difficult for the adversary to distinguish between legitimate and the dummy chunk. Random location was opted for storage of the legitimate chunks along with the dummy chunks stored in between them at the time of upload to the Google Docs. Every chunk is numbered to hide the original name of the chunk. Another distinguishing feature which the design provides is that the number associated with each chunk is changed every time the chunk is retrieved which makes it quite difficult for the adversary to trace the usage pattern of the chunks. The metadata files are used to store the information such as the file size, total number of legitimate and dummy chunks, the number denoting each chunk , etc. These metadata files are also encrypted which provides an additional security. At the time of download a random number of dummy chunks got downloaded along with actual chunks so that an adversary cannot identify actual file chunks.

The design offers the following basic functionality:

Login: The prime objective of this module is to perform authentication so that only authorized users are allowed to access the Google Docs. Login module authenticates the user by obtaining the user name and password from the user and verifies it with a Google account. If the

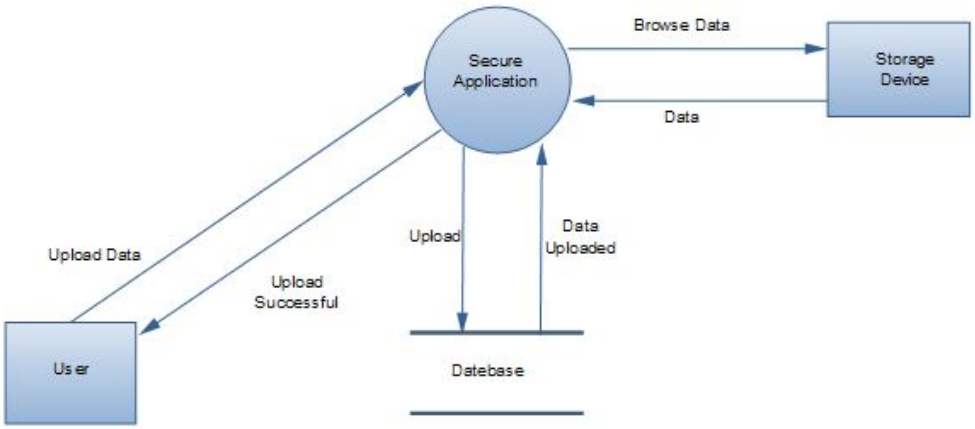
information is correct then only the application (DSSol) will proceed further else it will give authentication failure Fig 4.2.

Figure 4.2: Login



Upload: The upload module performs the task of encrypting the file. The encrypted file is then divided into chunks of equal size, which are then Xored to obtain subsequent chunks of equal size. The legitimate chunks and the Xored chunks are uploaded to the Google Docs. Additional dummy chunks are also uploaded with the Xored and legitimate chunks to secure them from traffic analysis. The information about all the uploaded chunks is stored in a text file, which is uploaded along with the chunks Fig 4.3.

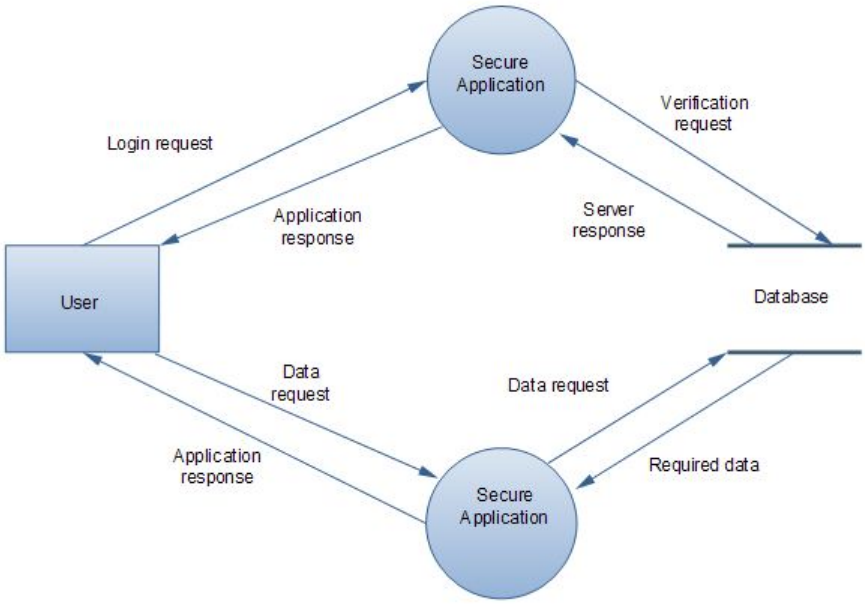
Figure 4.3: Upload



Download: The download process is just opposite of the upload process. The chunks are downloaded as per the details in the metadata files. Downloaded chunks include the original

chunks, the Xored chunks and some additional dummy chunks. These downloaded chunks are then used to regenerate the original file Fig 4.4.

Figure 4.4: Download



4.2 Implementation

The proposed design is focused to provide unobservability, robustness and traffic analysis resistant file system for the users to store their files in a cloud environment. Implementation of our design for Google Docs can be explained as follows:

4.2.1 Overview

The application is written in Python and does the following main tasks (explained in the subsequent section)

1. Format : Formats the Google Doc space for data upload.
2. Upload : Upload the user data.
3. Download : Download the user data.

The design components are detailed in the following sections along with the metadata structure required to support the stated functionality.

4.2.2 Metadata files

Metadata stores information that is used at various stages of our design. It helps in upload / download of document on Google Docs and related functionalities. Metadata files are encrypted with the passphrase generated by a password provided by the user and some random salt. These metadata files are encrypted and stored on the Google Docs. Different metadata files used in the design are as follows:

1. File info.txt: This metadata file contains the random code of length 4 bits for each file along with the corresponding file name. This file creates a mapping between the code generated and the original file name. The reason for having this 4 bit code is that whenever Xoring is performed, numerical code is used for the naming of generating files followed by chunk name1, further followed by chunk name2 and so on that can only be up to 255 characters as permitted by the operating system. For example, 4325_1x2. The numerical code thus generated hides the file name and information.
2. Info dummy.txt:-The file contains information about the dummy chunks that are uploaded to Google docs. It contains all the file names present in the Google docs with the status either the chunk is dummy one or the legitimate chunk represented by 0 and 1 respectively, hence helps in differentiating the legitimate file blocks from the dummy blocks .
3. Blockfile.txt:-Every time the chunks of a file were uploaded it replaces an equal number of dummy chunks already present on the Google docs. Blockfile.txt gives information about the dummy chunks, which are replaceing the dummy chunks already present on the Google

Docs and helps in identifying the location of legitimate blocks and no of dummy blocks left.

4. Info1.txt:-This metadata file stores the randomly generated code of length 4 along with the number of chunks in which the original file is divided and the key used for encryption of the file.

4.2.3 Logging Into Gmail Account

The execution starts from main() method and it accepts the username and password. After successful login, the user is given various options to choose from such as – class format, upload, download.

DocsSample: class

The object of this class is used for all interactions with any Gmail account. Once the main() function accepts the username and password for the Gmail account, it creates an object of the DocsSample class. From the constructor of this class the user is able to log in to Google Documents list feed service and Docs Client Service using the gdata library in Python.

Whenever the user starts using the method he is asked to choose among the following three options available 1:

1. Format (Choice 0)
2. Upload (Choice 1)
3. Download (Choice 2)

4.2.4 Formatting Google Docs Space

When the user selects choice 0 the format module is called. The prime objective of this module is to fill the whole Google Docs space with equal sized dummy files, which helps in hiding the legitimate files. These dummy blocks are replaced by the legitimate file chunks during the upload to Google Docs and are downloaded with legitimate chunks during the download. The system administrator who is looking at the files uploaded on the Google Docs cannot infer anything from the files on the Google Docs. Also, the file names are such that they mask the meaningful user file names. The file names of these dummy chunks are numbers that help in hiding the real file names over the cloud. The dummy chunks are filled with random / noise data for enhancing the security of the File System. A function FormatDocuments(self) are invoked to do all the above tasks. The info dummy.txt is updated accordingly to show the current file system status. This format fills the entire Google Docs space with numbered named file with dummy content which will be overwritten with the original file content when the user file is uploaded.

Algorithm 1 Choice List

```
input:choice (0 to 2)
if choice == 0 then
  Format Google Docs
else if choice ==1 then
  Upload file to Google Docs
else if Choice ==2 then
  Download a file from Google Docs
else
  Exit
end if
```

4.2.5 Uploading file to Google Docs

When the user selects choice 1 the upload module is called. The user is then asked to enter the complete file name and password.

During upload, the file to be uploaded is first checked within the entries in the info.txt file. If the file already exists in Google Docs a popup appears stating that the file already exists. In case if the file is being uploaded for the first time, the below mentioned procedure is followed: 2

1. A random code of length four is generated which is used for the processing of files. This code denotes the file number and is stored along with file name in info.txt file.

2. The file is then encrypted with AES encryption algorithm with a key generated randomly by M2Crypto library. The encrypted file is then divided into n legitimate chunks of equal size (256 KB each).

3. Xor operation is performed to implement the erasure codes 3.

4. The legitimate chunks along with the Xored chunks are converted to their equivalent base 64 encoded file and upload to Google Docs by replacing them with the available dummy blocks.

   The file info dummy.txt keeps the record of the dummy files in Google Docs which are replaced by the legitimate chunks.

5. A number of additional upload of dummy chunks also takes place to hide the legitimate chunks from the adversary. The number of dummy chunks uploaded from a file is equal to the number of legitimate chunks uploaded for that file to Google Docs. These additional chunks prevent an adversary from carrying out traffic analysis.

6. After the uploading all the blocks, uploading of encrypted metadata files will take place. The size of these metadata files will be same as the rest of the chunk.

Splitting a file 2

1. Calculate the size of the file which needs to be uploaded.
2. If required add padding at the end of the file, if size is not perfectly divisible by 256 (default chunk size).
3. Split the file into "n" equal number chunk where the size of each chunk will be equal to 256KB.

Erasure codes 3

In this technique of m-out-of-n file system is used, where :

m is the total number of chunks of original file of size 256 KB obtained after padding.

If the number of chunks obtained after padding are odd then an additional default chunk of 256KB is added to m to make the total number of chunks even.

n is a parameter which is dependent on m and is obtained as follows:

divide m(always even) by 4,

n= 2m if m is divisible 4,

n =2m+3 if m is not divisible by 4

The implementation of the upload module can be represented by the following algorithms:

Algorithm 2 Algorithm for upload

```
input:filename
Generate randon code of length four.
Append code into info.txt.
Encrypt the file.
Split the file into equal chunks.
size= file size
if size %n == 0 then
  chunk =n
else
  Append padding
  chunk =n
end if
Xoring of chunks.
Convert all files into its equivalent base 64 encoded files.
for i = 1 to n do
  num = randomly select a number from 0 to 1.
  if num==0 then
    Upload a random chunk.
  else
    Upload a legitimate chunk.
  end if
end for
```

Algorithm 3 Xoring of chunks

m = number of legitimate chunk formed from the file.
n = total number of chunks to be uploaded.
if m %4 ==0 **then**
  n =2m
else
  n = 2m +3
end if
for $i = 1$ to m/2 **do**
  XOR the chunks in groups of 2 that is m/2 more chunks are obtained by calling XorFunction() method.
end for
for $i = 1$ to m/2 **do**
  XOR the chunks in group of two by using the alternately numbered chunk by calling XorFunction() method.
end for
if m%4 != 0 **then**
  XOR the left chunks with the complement of each other by calling XorFunction() method and update Blockfile.txt with the Xor files names and it's corresponding dummy chunk with which it is replaced.
end if

4.2.6 Downloading file from Google Docs

When the user selects choice 2 the download module is called. The user is then asked for the filename and the password.

The details of the file requested by the user is searches in the list of files maintained in the info.txt to validate whether it exists in the Google file system or not.

If the file doesn‘t exist in Google Docs, a popup appears stating that the file doesn‘t exists. If it exists, all the file chunks are identified from the info1.txt and they are downloaded using the Gdata APIs.

Once all the chunks are downloaded belonging to a particular file, we regenerate the original file using the below algorithm: 4

Algorithm 4 Algorithm for download

```
if all chunks are downloaded successfully then
  Combine them and decrypt the file.
else
  Identify the missing chunks.
  Try to make the missing chunks from the available consecutive Xored chunks and append
  the generated chunk to list of available chunks.
  if required chunks == list of chunks available then
    Combine them and decrypt the file.
  else
    Try to make the missing chunks from the available alternatively Xored chunks and append
    the generated chunk to list of available chunks.
    if required chunks == list of chunks available then
      Combine them and decrypt the file.
    else
      File cannot be retrieved.
    end if
  end if
end if
```

Chapter 5

Analysis

We performed the analysis based on the following parameters to measure the performance of the proposed design:

1. Average time taken to upload / download a file of different sizes.

For calculating the average upload / download time, 100 files of each file size ranging between 32 KB to 5 MB are processed (uploaded / downloaded) and the time obtained is represented as a graph.

The internet speed was in the range of

Upload Speed: 0.55 to 0.65 Mbps

Download Speed: 0.35 to 0.50 Mbps

5.1 Table representing average upload / download time

The arithmetic mean of all the obtained time values for a particular file size give the average time of upload / download for that file Table 5.1.

File Size	Average Upload Time (in minutes)	Average Download Time (in minutes)
32KB	5.607	3.75
64KB	4.95	3.812
128KB	5.518	3.589
256KB	5.531	3.72
512KB	6.46	4.571
1MB	8.27	6.5
2MB	9.021	7.072
5MB	16.08	11.131

Table 5.1: Variation of average download / upload time with respect to the file size.

The variation of time with respect to file size Fig 5.1 Fig 5.2

Figure 5.1: Variation of upload time with file size

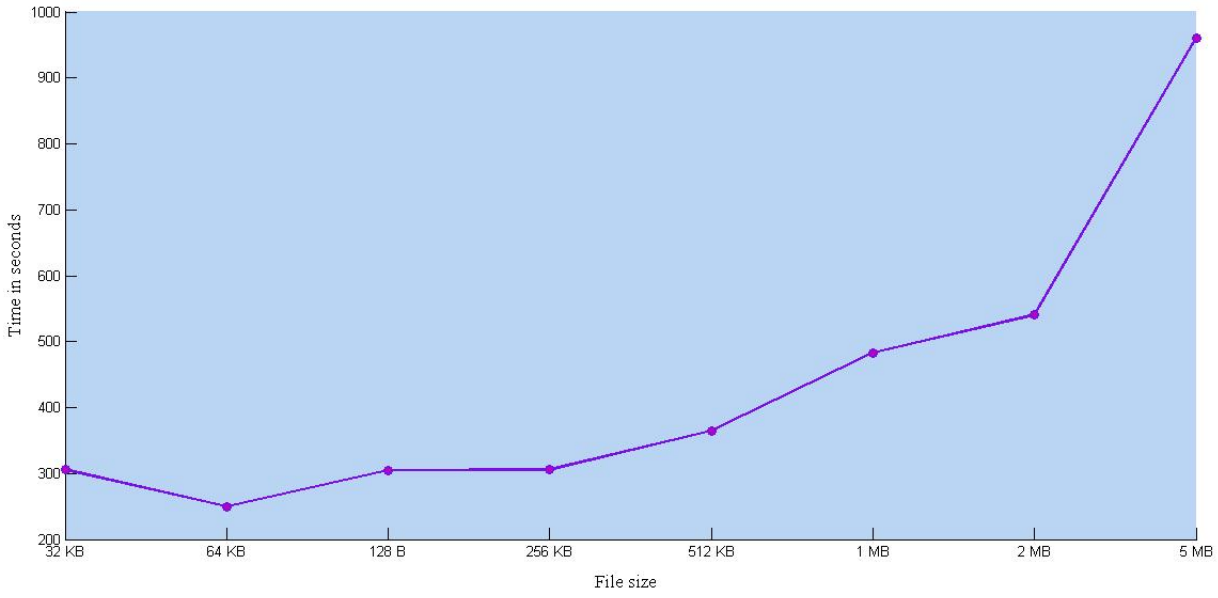
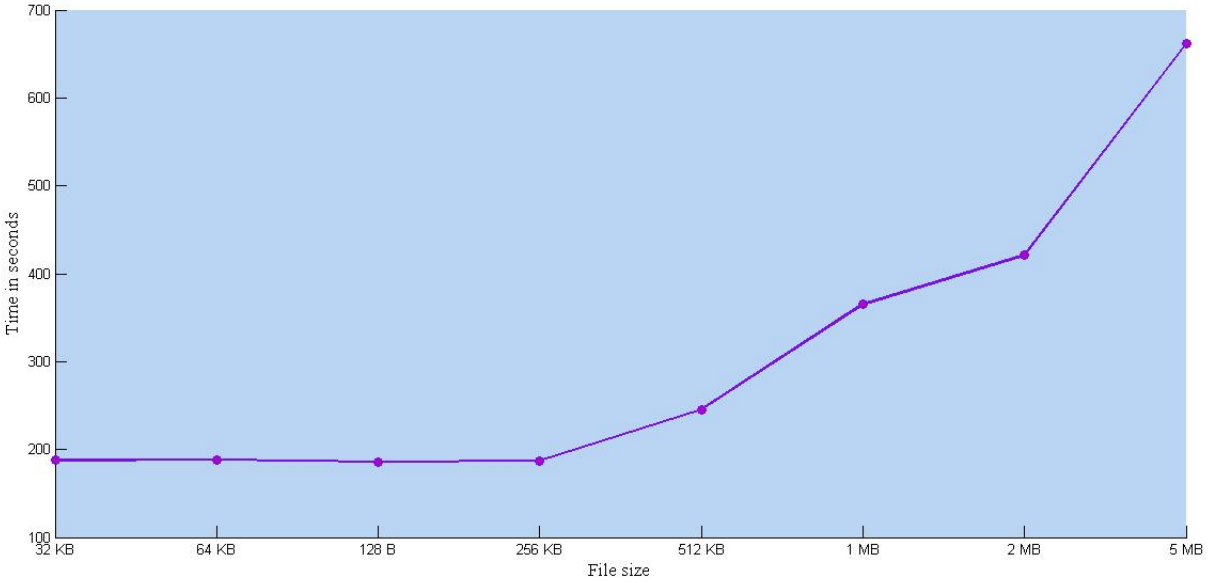


Figure 5.2: Variation of download time with file size



5.1.1 Efficiency of algorithm of erasure coding

Whenever a user wants to download a file from the Google Docs, he will specify the name of the file to the application. Then the application tries to downloading all the legitimate and Xored chunk present on the Google Docs. If all the legitimate chunks are downloaded successfully, then the file can be retrieved with 100% probability. But as the Google itself is an adversary, there might exist a situation where Google deletes some of the file chunks from the Google Docs randomly. This might result in the deletion of the legitimate chunks for a file and hence the file could not be regenerated successfully. In such situation Tornado codes helps in retrieving the file.

The proposed design follows,m out of n method where m is the number of legitimate chunks of the original file and n are the total number of chunks uploaded (legitimate and Xored chunks). According to the method if we randomly take any m chunks out of n chunks, we should be able to regenerate the original file. This provides the basis for calculating the efficiency of our method. In the present application the range of the successfully downloaded chunks may vary from 0 to n. The efficiency of the Tornado codes can be calculated as the probability of regenerating the original file from any random m chunks downloaded successfully out of the n chunks uploaded to the Google Docs. There might be cases where:

m <= Successfully downloaded chunks < n

In considering such test cases where at least m out to total uploaded chunks of a file can be successfully downloaded , we analyzed that in about 80% test cases we were able to regenerate the original file back.

5.2 Security analysis

The objective of this design is to have a Robust and Traffic Analysis Resistant Cloud File System against an adversary who has the capability to monitor and learn traffic patterns, can identify the file sizes and number of accesses to a particular file. This section explains how the proposed design fulfills the stated requirements.

1. Sequence Uncertainty: A sequence can be defined as a series of chunk transactions made to the server. So the sequence certainty is to identify which set of chunks in a transaction belongs to a file on the disk. The objective of the adversary is to identify the legitimate number of chunks from the whole set of uploaded chunks (legitimate chunks of the file and additional uploads of dummy chunk). In the proposed design, the number of dummy chunk and legitimate chunk are kept equal therefore the time of upload and download the chunk to be accessed is selected at random and can be either a legitimate chunk or a dummy chunk. This way the sequence of access of chunks from the Google Docs will not reveal any information regarding which is the dummy chunk or the legitimate chunk. So, if the adversary is analyzing the traffic he cannot distinguish between legitimate chunk and the dummy chunk uploaded.

   For a given transaction the probability the of identifying the legitimate chunks from the set of chunk being accessed is $1/^{x+n}C_n$

   where x is the additional number of chunks being accessed.

   and n is the legitimate chunk of the file.

   So, the probability of identifying one chunk correctly from x+n chunks is $1/x + n$.

   Subsequently, the probability of identifying the next chunk will be 1/x+n-1.

   The overall probability will be:

   $(1/x + n) + (1/x + n - 1) + (1/x + n - 2) + (1/x + n - 3) + ...$

   In summation we get:

   $\sum$ 1/x+n-z

   Higher the value of "n" which is legitimate chunk, higher will be the value of x which decreases the probability of identifying the legitimate chunks and hence greater will be the security. This will also increase the complexity for the attacker as he does not know the value of n for the file that is uploaded.

2. Location Uncertainty: Location uncertainty is the complexity of locating a specific file in the given storage space. For an attack to be successful the attacker wants to identify the location where chunks belonging to a particular file reside on the Google Docs. In the proposed design the uploaded chunks randomly replace the dummy chunks already available on the Google Docs. Thus, location Uncertainty provides unobservability by increasing the complexity of locating a legitimate chunk on the Google Docs.

Whenever the Google Docs are accessed the chunk location is appended to the top of the files already uploaded to Google Docs along with a time stamp. So, the location and time stamp reveals information associated with a file that is which file has been accessed most recently and at what time. The proposed design does not tamper the time stamp but selects a location randomly to store a file from the available locations on the Google Docs. From the time stamp it can be easily inferred that which file is being accessed. This decreases the sample space to be targeted by an attacker, who can target a set of most recently accessed files and try to identify the legitimate chunks of a file. Even after such an attack the probability of guessing that whether a given chunk is a legitimate chunk or a dummy chunk is 1/2.

3. Proposed design has used AES encryption whose computational complexity to recover the key is $2^{189.7}$. AES encryption is used in two places one when files are uploaded to Google Docs, before uploading they are encrypted and the other for the encryption of metadata files using password based encryption.

4. The proposed design use password based encryption to provide an extended security. The user is required to choose an additional password other than the gmail account password. This additional password along with a salt is used to generate a key, which is used for the encryption of metadata files. This ensures that even if the user selects a weak password the key generated from the password will be strong and hence strengthens the security of the system.

5. Association Uncertainty: Till now we were considering the attacks valid for a single transaction to the Google Docs but there might be a case where an adversary is performing the attack by observing multiple transactions. Association certainty can be defined as the capability of identifying the legitimate chunks of a file across multiple transaction of a file to Google Docs. To perform this attack one needs to perform intersection between multiple transactions. This kind of attack will be useful when the user is downloading a file from the Google Docs. The attacker can observe the files that are being downloaded. If a user is downloading the same file multiple number of times then the intersection attack will reveal the legitimate chunks of a particular file and their number. We will address this attack in our future goals.

5.3 Future scope (Limitation)

The design has been developed as a client machine based application which provides protection against the adversary tracking important information by monitoring the data traffic and usage patterns. It is useful for data security which has become a prime concern for both business as well as for the end user and can be extended further as:

1. Though the design has been developed as a client machine based application, but it can be used in a browser as a browser plugin.
2. The user needs to remember his additional password used for encryption always, as it cannot be recovered if the user forgets it. The scheme can be improved to recover the lost password.
3. The count of the dummy chunks uploaded /downloaded along with the file chunks is equal to the sum of legitimate and Xored chunks i.e. n, but the count of dummy chunks can be increased to be 2n, 3n, etc which will considerably decrease the probability of identifying legitimate chunks and hence increase the security of the application. But this will affect the utility of the application as the total time required for uploading and downloading the file will be increased in proportion to the increase in the number of dummy chunks.
4. It can also be extended to other services like Drop Box.
5. It can also be used for different file formats like spreadsheets, pdfs , presentations and even for multimedia files.

5.4 Conclusion

In recent times cloud has become the favorite choice of the users for storing the data over the Internet. In the present design we present a solution for the security of data stored over a cloud environment against an adversary which can be the system administrator himself. The design provides protection against attacks possible due to monitoring traffic analysis and study of usage patterns of the files, thereby providing unobservability of the important data.

Chapter 6

Appendix

6.1 Requirements

Hardware Requirements

1. Basic hardware requirements (Recommended): 512 MB RAM.
2. Pentium IV or above processor Ethernet cards (For for internet connectivity)

Software Requirements

1. Basic software requirements: Operating System: Ubuntu or any other Linux Distribution,
2. Windows XP or later
3. Python 2.6 or later
4. gdata and atom libraries may have to be explicitly installed
5. Pentium IV or above processor Ethernet cards (For for internet connectivity)

6.2 User Manual

Pre Requisite:

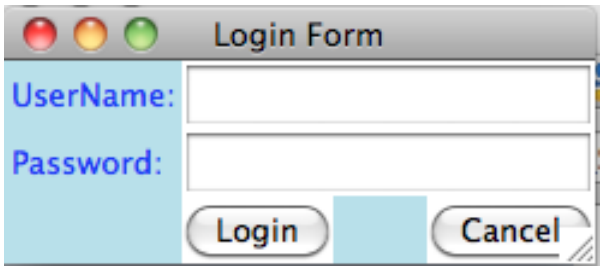
Install following dependency in a common folder:

1. gdata python client library `https://developers.google.com/gdata/articles/python_client_lib`
2. M2Crypto `http://chandlerproject.org/Projects/MeTooCrypto`
3. Sympy `http://code.google.com/p/sympy/wiki/DownloadInstallation`
4. Passlib `http://packages.python.org/passlib/install.html#`

Please follow the mentioned steps: The downloaded folder contains the login.py file.

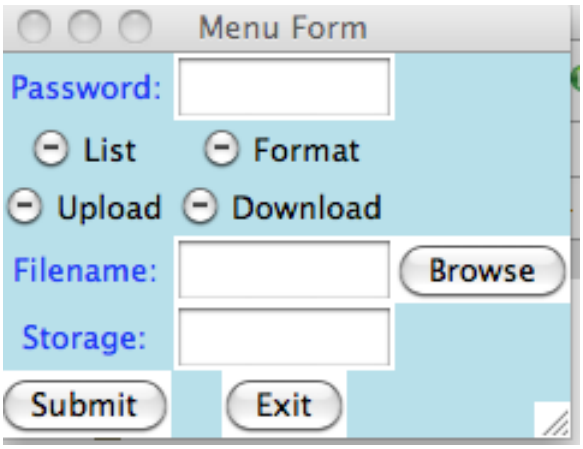
1. Move to the location of the downloaded folder from terminal/command prompt and run login.py by typing the command pyhton login.py.
2. A window will pop up. Enter your gmail username and password Fig 6.1

Figure 6.1: Login



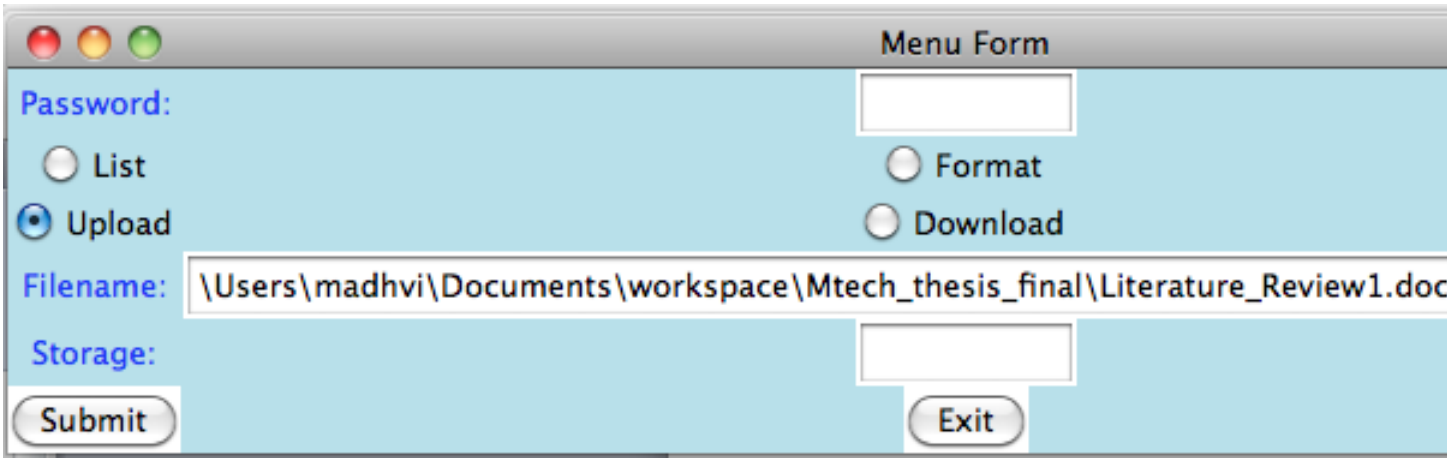
3. After the successful authentication another window will appear which will ask you for selecting other password required for data security and a number of options: Fig 6.2
   (a) List
   (b) Format
   (c) Upload
   (d) Download

Figure 6.2: Options



4. When uploading for the first time select format option and specify the amount of storage you want to use on Google Docs. Then select the upload option with the filename you want to upload (you can also browse file to be uploaded). Fig 6.3

Figure 6.3: Upload



In a similar way you can download the file by specifying the filename and password selected before.

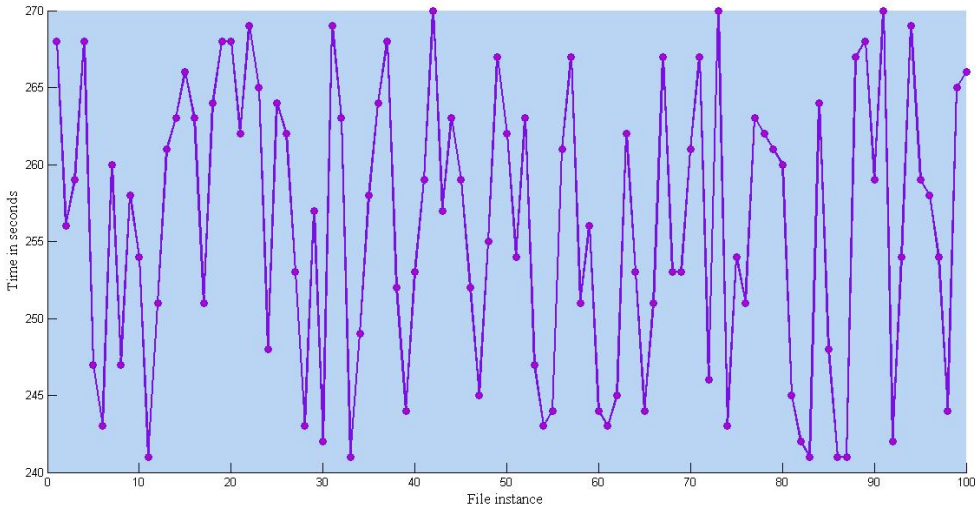
Note: Please remember the additional password selected as you will be requiring this password for uploading and downloading the files to Google docs.

6.3 Average time for uploading and downloading a file

6.3.1 Graphs for uploading a file

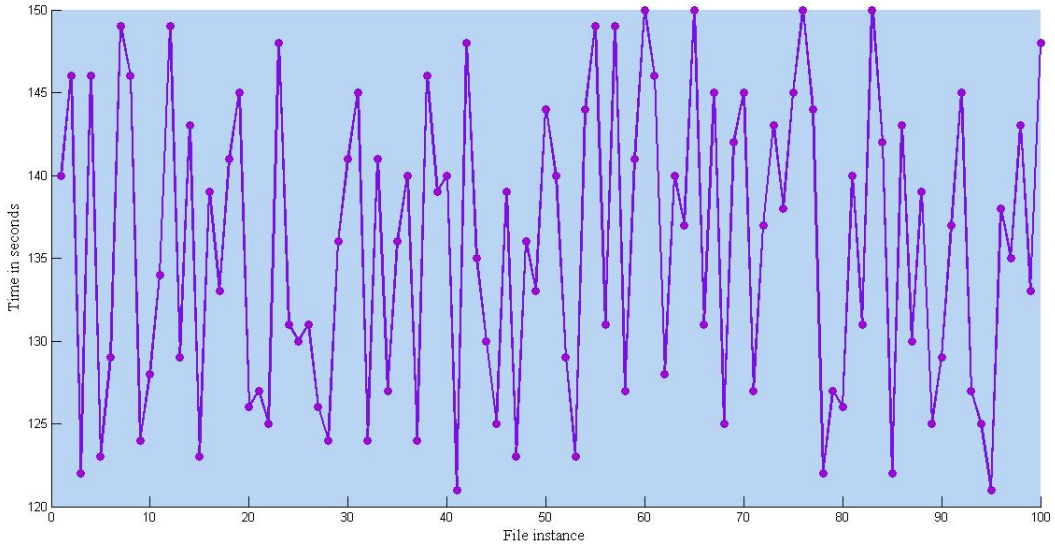
1. File Size = 32kb Fig 6.4

Figure 6.4: Upload Time variation for 32 kb file.



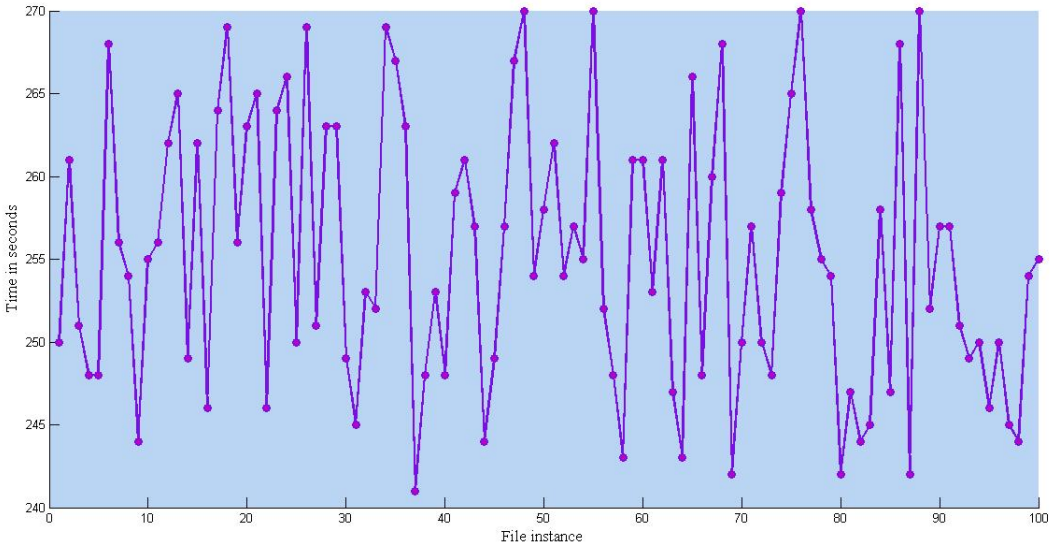
2. File Size = 64kb Fig 6.5

Figure 6.5: Upload Time variation for 64 kb file.



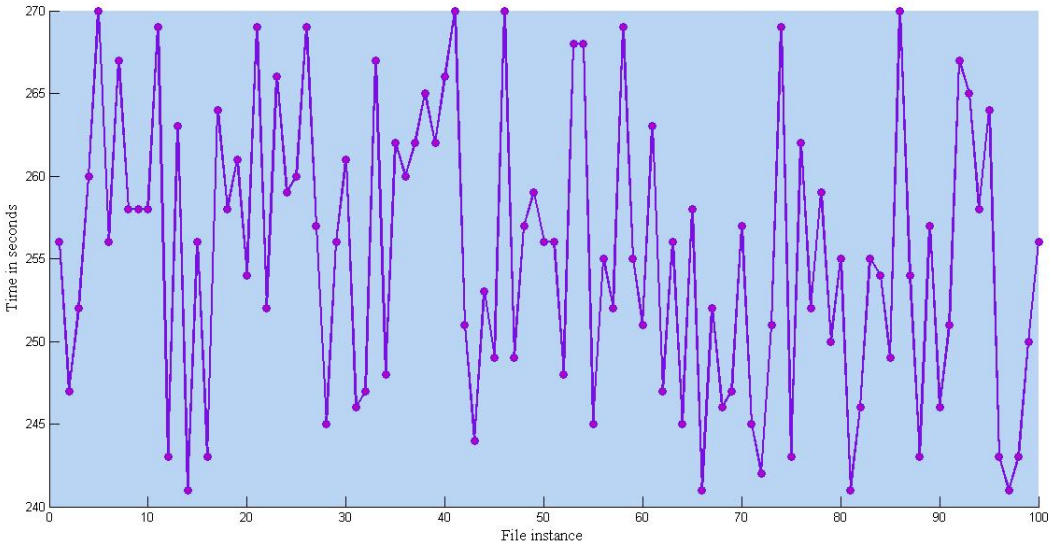
3. File Size = 128kb Fig 6.6

Figure 6.6: Upload Time variation for 128 kb file.



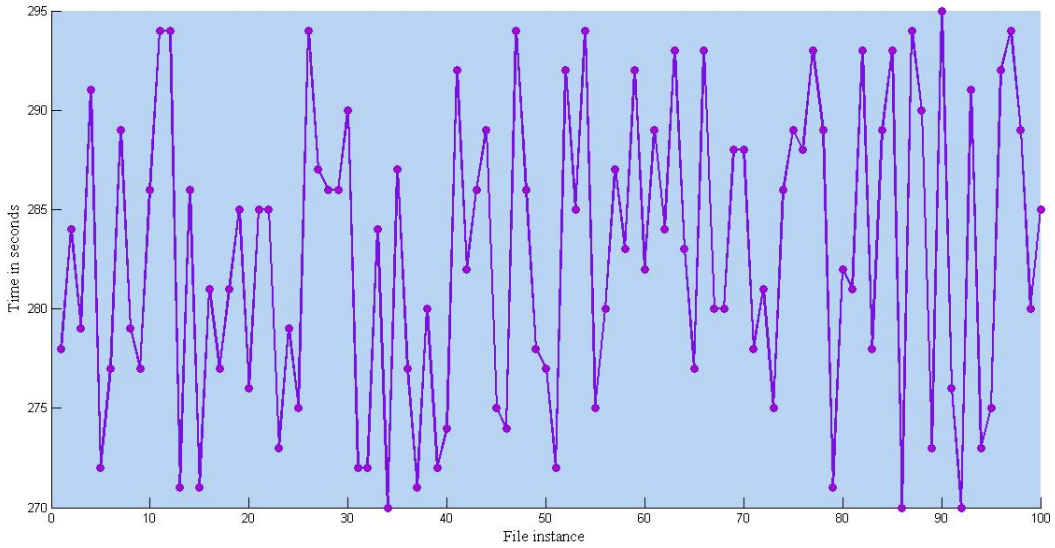
4. File Size = 256kb Fig 6.7

Figure 6.7: Upload Time variation for 256 kb file.



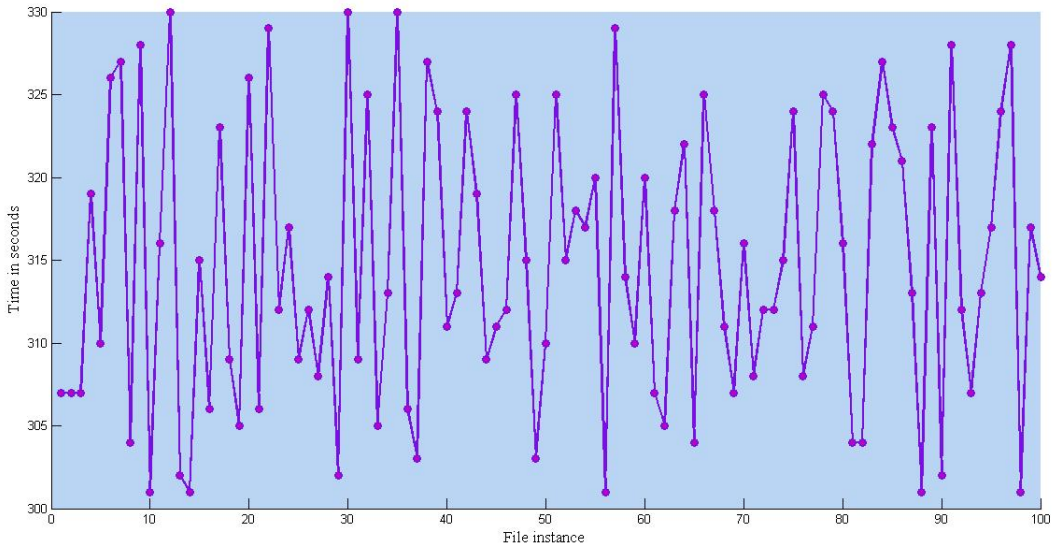
5. File Size = 512kb Fig 6.8

Figure 6.8: Upload Time variation for 512 kb file.



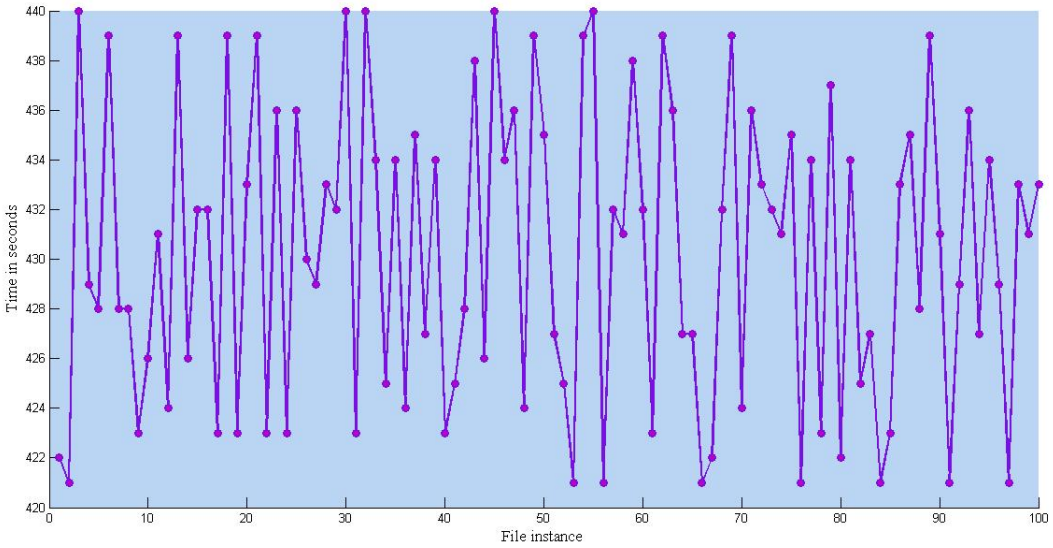
6. File Size = 1Mb Fig 6.9

Figure 6.9: Upload Time variation for 1Mb file.



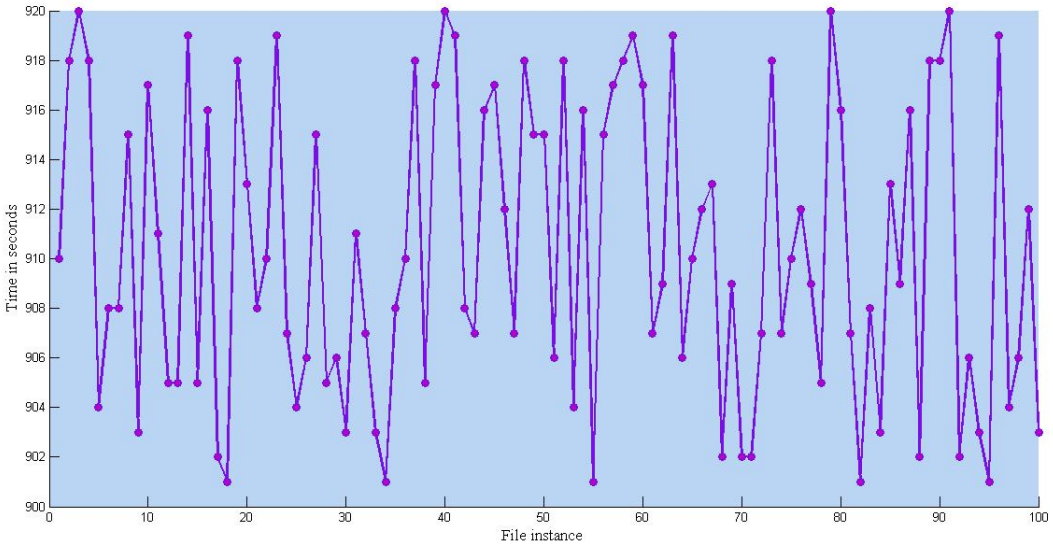
7. File Size = 2Mb Fig 6.10

Figure 6.10: Upload Time variation for 2Mb file.



8. File Size = 5Mb Fig 6.11

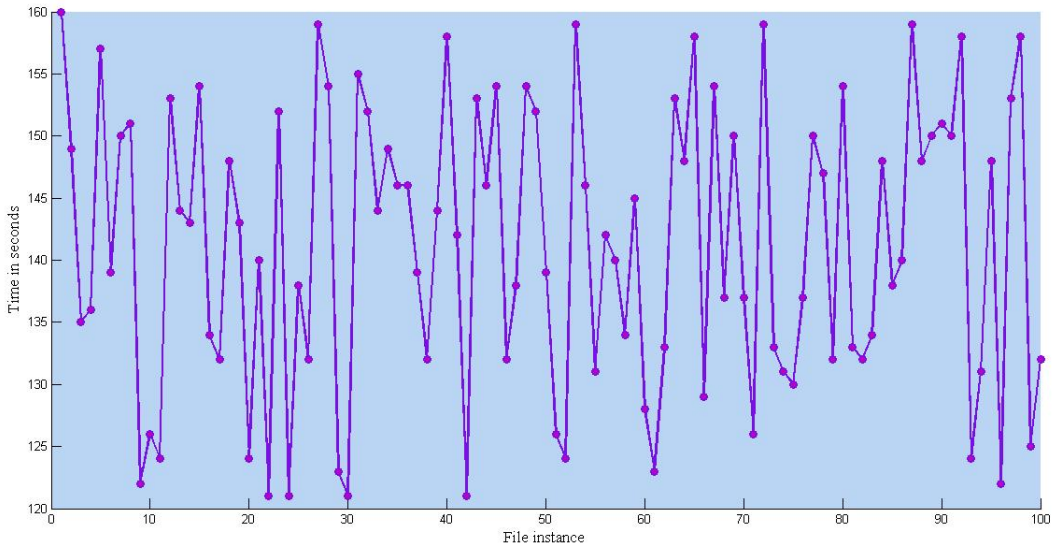
Figure 6.11: Upload Time variation for 5Mb file.



6.3.2 Graphs for downloading a file

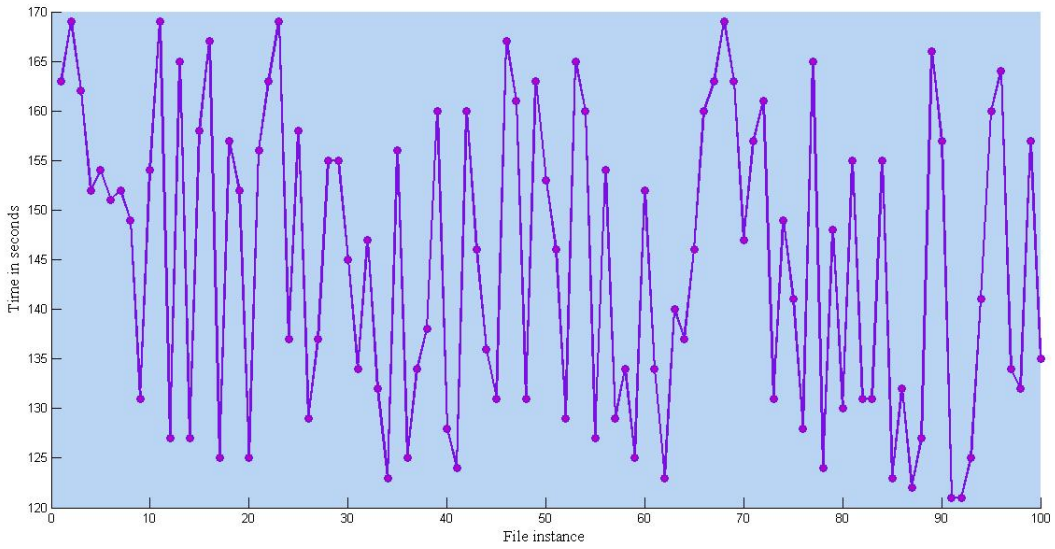
1. File Size = 32kb Fig 6.12

Figure 6.12: Download Time variation for 32 kb file.



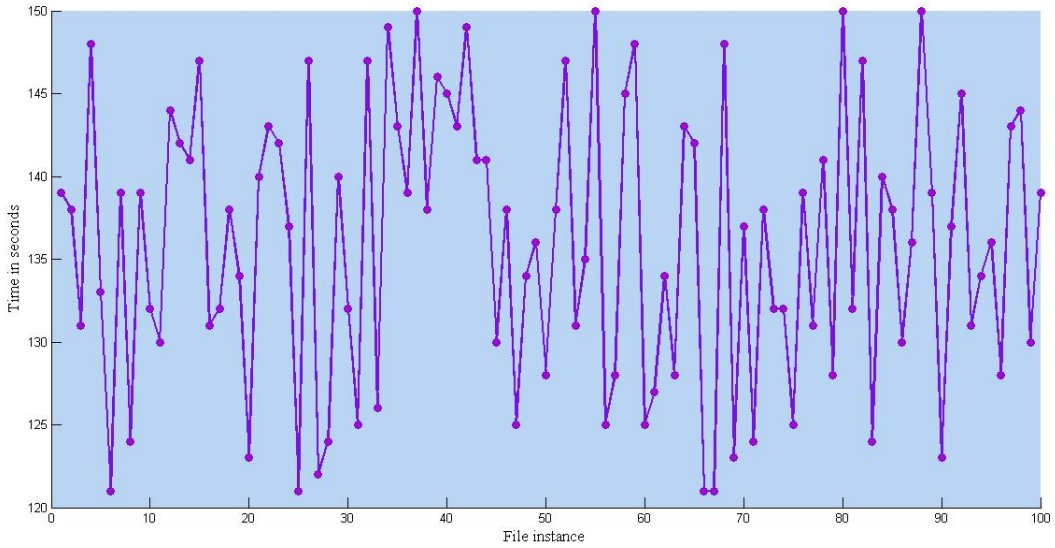
2. File Size = 64kb Fig 6.13

Figure 6.13: Download Time variation for 64 kb file.



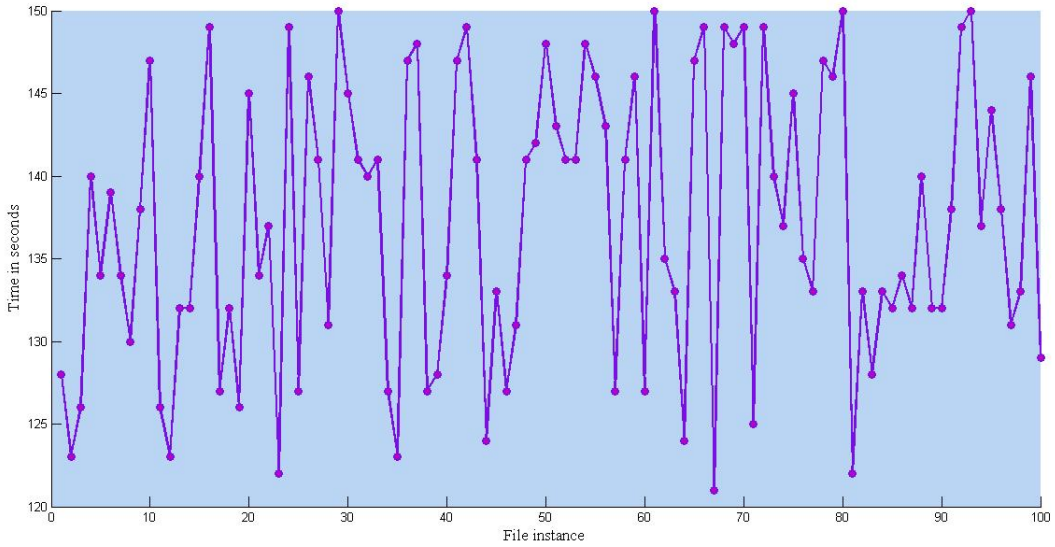
3. File Size = 128kb Fig 6.14

Figure 6.14: Download Time variation for 128 kb file.



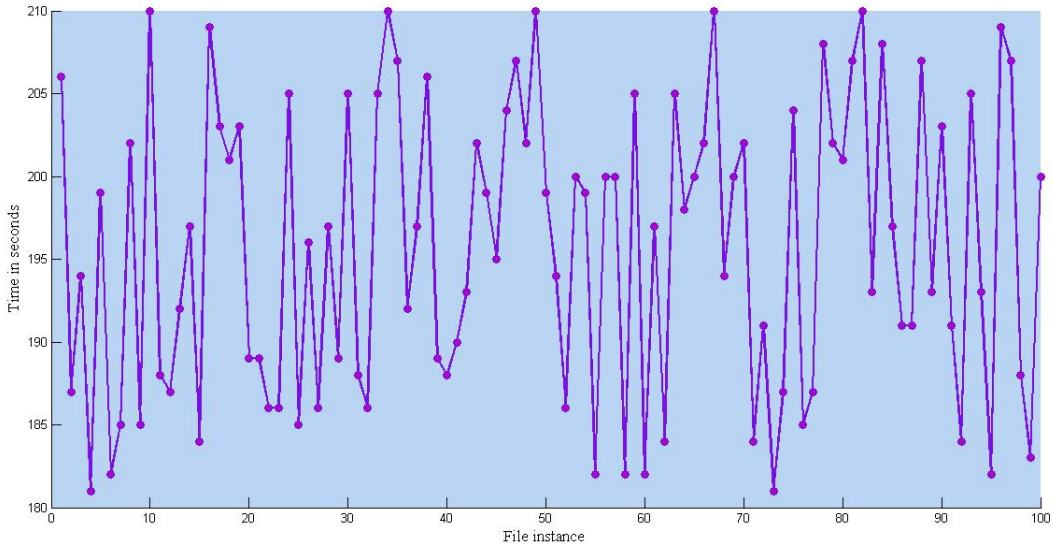
4. File Size = 256kb Fig 6.15

Figure 6.15: Download Time variation for 256 kb file.



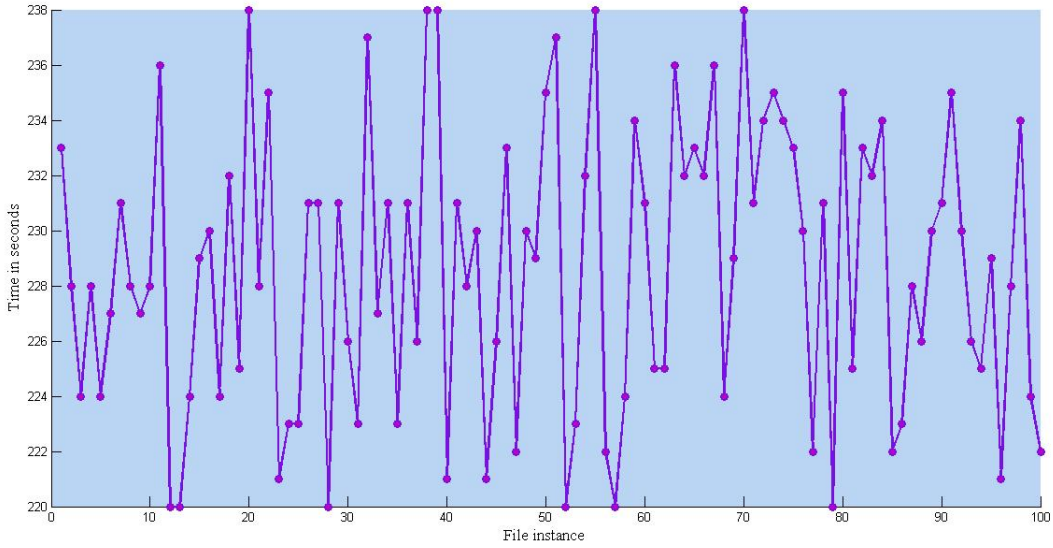
5. File Size = 512kb Fig 6.16

Figure 6.16: Download Time variation for 512 kb file.



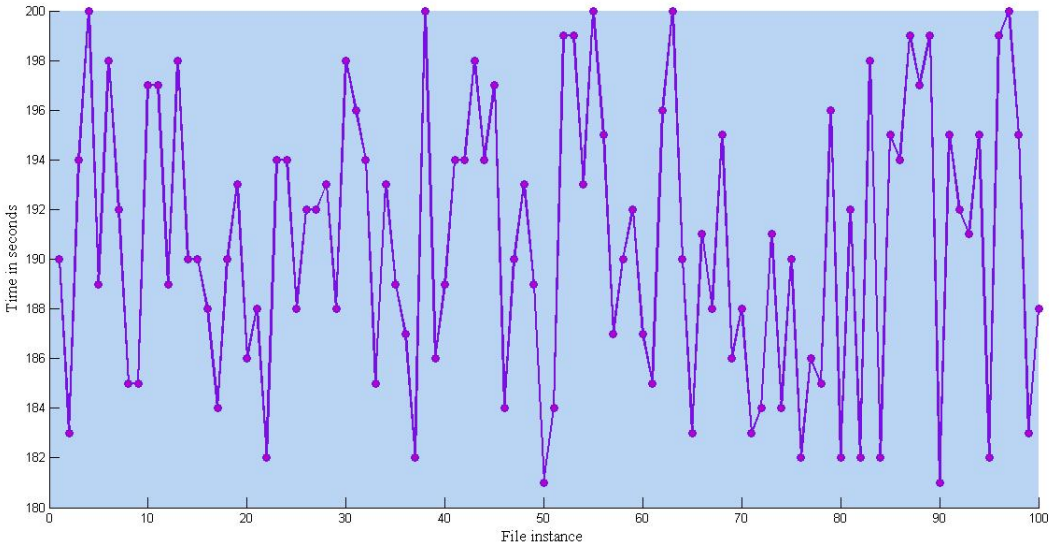
6. File Size = 1Mb Fig 6.17

Figure 6.17: Download Time variation for 1Mb file.



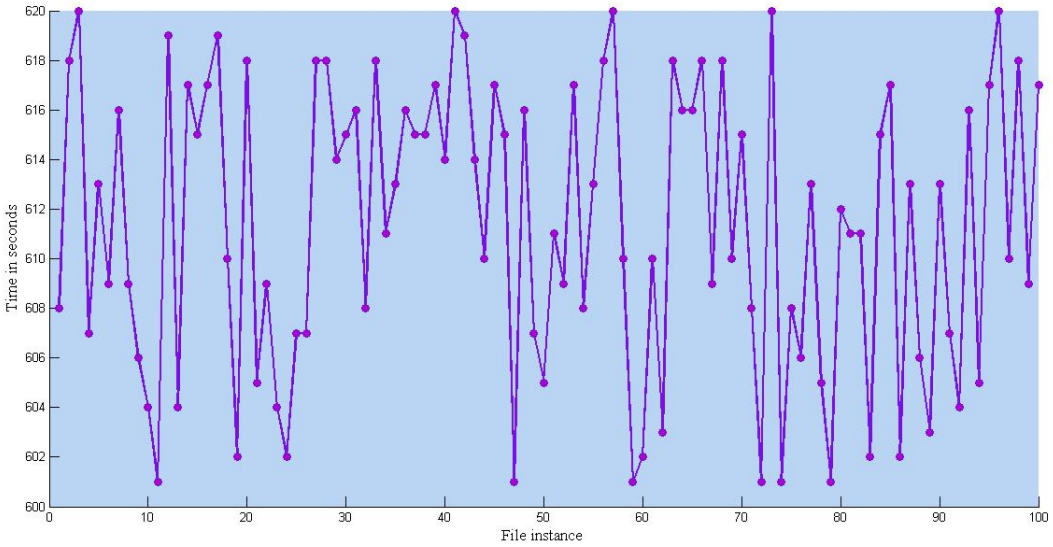
7. File Size = 2Mb Fig 6.18

Figure 6.18: Download Time variation for 2Mb file.



8. File Size = 5Mb Fig 6.19

Figure 6.19: Download Time variation for 5Mb file.



Bibliography

[1] Chandrasekhar, Bharath. Did Amazons Aggressive Algorithms Prevent Customer Data Loss Trend Cloud Security.

[2] Alpeyev, Pavel, Joseph Galante, and Mariko Yasu, Amazon.com Server Said to Have Been Used in Sony Attack, Bloomberg News.

[3] Mohamed Al Morsy, John Grundy and Ingo Mller Analysis of The Cloud Computing Security Problem.

[4] F. A. Alvi1, B.S Choudary2, N. Jaferry3, E.Pathan4:A review on cloud computing security issues & challanges.

[5] Fei Hu1, Meikang Qiu2, Jiayin Li2, Travis Grant1, Draw Tylor1, Seth McCaleb1, Lee Butler1 and Richard Hamner 1 A Review on Cloud Computing: Design Challenges in Architecture and Security.

[6] Wayne Jansen,Timothy Grance Guidelines on Security and Privacy in Public Cloud Computing.

[7] Trend Micro Incorporated(TYO:4704;TSE:4704)Security Threats To Evolving Data Centers.

[8] Alexander Cockburn and Jeffrey St. Clair Counter Punch.

[9] A Framework for the Analysis of MixBased Steganographic File Systems Claudia Diaz, Carmela Troncoso, and Bart Preneel.

[10] Anderson, R.J., Needham, R.M., Shamir, A.: The steganographic file system. In: Aucsmith, D. (ed.) IH 1998. LNCS, vol. 1525, pp. 7382.Springer, Heidelberg (1998).

[11] Carl van Schaik, Paul Smeddle: A Steganographic File System Implementation for Linux, University of Cape Town, South Africa, October 1998.

[12] McDonald, A.D., Kuhn, M.G.: StegFS: A steganographic file system for linux. In:Pfitzmann, (ed.) IH 1999. LNCS, vol. 1768, pp. 462477. Springer, Heidelberg(2000).

[13] Pang, H., Tan, K.-L., Zhou, X.: StegFS: A steganographic file system. In: Proceedings of the 19th International Conference on Data Engineering, pp. 657667. IEEEComputer Society Press, Los Alamitos (2003).

[14] Hand, S., Roscoe, T.: Mnemosyne: Peer-to-peer steganographic storage. In: Druschel,P., Kaashoek, M.F., Rowstron, A. (eds.) IPTPS 2002. LNCS, vol. 2429, pp.

[15] Giefer, C., Letchner, J.: Mojitos: A distributed steganographic file system. Technical report, University of Washington (2004).

[16] Jammalamadaka R.C. , Gamboni R, Mehrotra S, Kent E. Seamons, Venkatasubramanian N: gvault : A Gmail Cryptoraphic Network file System, University of California, Irviney, Brigham Young Universityz, University of Bologna,Italy.

[17] Luby, M. G., M. Mitzenmacher, M.A. Shokrollahi, and D.A. Spielman, Efficient Erasure Correcting Codes, IEEE Transactions on Information Theory, 47(2), 569-584, February 2001.

[18] Troncoso, C., Diaz, C., Dunkelman, O., Preneel, B.: Traffc analysis attacks on a continuously-observable steganographic file system. In: Furon, T., et al. (eds.) IH 2007. LNCS, vol. 4567, pp. 220236. Springer, Heidelberg (2008).

[19] Zhou, X., Pang, H., Tan, K.-L.: Hiding data accesses in steganographic file system. In: Proceedings of the 20th International Conference on Data Engineering, pp.572583. IEEE Computer Society, Los Alamitos (2004)

[20] . G. Ateniese, R. Burns, R. Curtmola, J. Herring, L. Kissner, Z. Peterson, and D. Song. Provable data possession at untrusted stores. In CCS, pp. 598609, 2007.

[21] . Y. Dodis, S. Vadhan, and D. Wichs. Proofs of retrievability via hardness amplification. In TCC, pp. 109127, 2009.

[22] A. Juels and B. S. Kaliski. PORs: Proofs of retrievability for large files. In CCS, pp. 584597, 2007.

[23] H. Shacham and B. Waters. Compact proofs of retrievability. In ASIACRYPT, pp. 90107, 2008.

[24] M. T. Goodrich, R. Tamassia, and A. Schwerin. Implementation of an authenticated dictionary with skip lists and commutative hashing. In DISCEX II, pp. 6882, 2001.

[25] C. Chris Erway,Alptekin Kp, Charalampos Papamanthou, Roberto Tamassia. Dynamic Provable Data Possession.

[26] J. L. Hafner. WEAVER Codes: Highly fault tolerant erasure codes for storage systems. In FAST-2005: 4th Usenix Conference on File and Storage Technologies, San Francisco, December 2005.

[27] G. Ateniese, R. D. Pietro, L. V. Mancini, and G. Tsudik. Scalable and effcient provable data possession. In SecureComm, pp. 110, 2008.

[28] . Kevin D. Bowers, Ari Juels RSA,Alina Oprea. HAIL: A High-Availability and Integrity Layer for Cloud Storage.

[29] Thomas Ristenpart, Eran Tromer, Hovav Shacham, Stefan Savage. Hey, You, Get Off of My Cloud: Exploring Information Leakage in Third-Party Compute Clouds.

[30] . Erasure Coding vs. Replication: A Quantitative Comparison by Hakim Weather-spoon and John D. Kubiatowicz Computer Science Division.

[31] James S. Plank, A Tutorial on Reed-Solomon Coding for Fault Tolerance in RAID-like systems.

[32] KHISTI A. Tornado codes and Luby Transform Codes[EB/OL]. October, 2003.

[33] Insider Threats Related to Cloud Computing–Installment 2: The Rogue Administrator `http://www.cert.org/blogs/insider_threat/2012/08/title_insider_threats_related_to_cloud_computing--installment_2_the_rogue_administrator.html`

[34] Cloud Security Alliance, Top Threat to Cloud Computing V1.0, March 2010. [Online]. Available: https://cloudsecurityalliance.org/topthreats/csathreats.v1.0.pdf

[35] D. Takahashi, French hacker who leaked Twitter documents to TechCrunch is busted, March 2010. [Online]. Available: http://venturebeat.com/2010/03/24/french-hacker-who- leaked-twitter-documents-to-techcrunch-is-busted/