SECURE STORAGE OF CLOUD DATA BY AES ENCRYPTION IN XTS MODE

A PROJECT REPORT

Submitted By

PERIYASAMY A (07C76)
SARVESH GHAUTHAM M K (07C99)
SOWMIYA NARAYAN S (07C108)

Guided By

Dr. S. MERCY SHALINIE

in partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

THIAGARAJAR COLLEGE OF ENGINEERING

(An autonomous institution affiliated to Anna University)

(An ISO 9001:2008 Certified Institution)

APRIL 2011

THIAGARAJAR COLLEGE OF ENGINEERING

(An autonomous institution affiliated to Anna University)

(An ISO 9001:2008 Certified Institution)

BONAFIDE CERTIFICATE

Certified that this project report **SECURE STORAGE OF CLOUD DATA BY AES ENCRYPTION IN XTS MODE** is a bonafide work of **PERIYASAMY A**, **SARVESH GHAUTHAM M K** and **SOWMIYA NARAYAN S** who carried out the project work under my supervision.

SIGNATURE
Dr. S. Mercy Shalinie,
Head of the Department,
Department of CSE,
Thiagarajar College of
Engineering, Madurai-15.

SIGNATURE
Dr. S. Mercy Shalinie,
Head of the Department,
Department of CSE,
Thiagarajar College of
Engineering, Madurai-15.

Station: Madurai

Date:

Submitted for the "VIVA-VOCE" examination held at Thiagarajar College of Engineering on _____.

INTERNAL EXAMINER

EXTERNAL EXAMINER

Acknowledgement

We wish to express our deep sense of gratitude to **Dr. V. ABHAI KUMAR**, Principal, Thiagarajar College of Engineering for his support and encouragement throughout this project work.

We wish to express our sincere thanks to **Dr.S.Mercy Shalinie**, Head of the Department of Computer Science and Engineering for her support and ardent guidance.

We owe our special thanks and gratitude to **Mrs.G.Sujitha**, Research Associate, Department of Computer Science and Engineering for her guidance and support throughout our project.

We are also indebted to all the teaching and non-teaching staff members of our college for helping us directly or indirectly by all means throughout the course of our study and project work.

We thank our parents, family members and friends for their moral support and encouragement for our project.

Abstract

The rising abuse of information stored on large data centers in cloud emphasizes the need to safeguard data. Despite adopting strict authentication policies for cloud users and transferring data over SSL/TLS when data reaches data center it is vulnerable to numerous attacks. Protecting stored data effectively involves embedding hardware wherein encryption is performed on disk. The other interesting and most widely adaptable methodology is safeguarding at software level. Encrypting data at rest prevents unauthorized access of confidential information. Encryption of large data which is a time consuming process needs to be controlled by an efficient application of the process in parallel mode. The aim of this project is to propose a method to perform encryption in parallel using standard XTS-AES approach. The proposed methodology gives efficient results when compression is performed before encryption.The experimental results generated gives compression ratio in the order of 1:10 which proves that the concept suits well for a cloud computing data-storage environment.The above concept implemented using MapReduce framework shows that vendor specific cloud computing model can make use of such concept for securing their data store.

Contents

1 Introduction **7**

1.1 Cloud Storage as a Service 7

1.2 Map Reduce 7

1.2.1 Hadoop's MapReduce 9

1.2.2 Fault Tolerance 11

1.2.3 Worker Failure 11

1.2.4 Master Failure 11

1.3 Need for Security in Clouds 11

2 Problem Definition **13**

2.1 Existing Approaches 13

2.2 Problem Statement 15

2.3 Proposed Solution 15

3 System Requirements **16**

3.1 Hardware Specification 16

3.2 Software Specification 16

4 Key Management and Encryption **17**

4.1 Security of Keys 17

4.2 Secure Key Generation 18

4.2.1 Algorithm for Key Generation 18

4.2.2 Key Management 19

4.3 Encryption 20
4.3.1 Map Phase 21
4.3.2 Reduce Phase 21
4.3.3 Modes of Encryption 22
4.4 Compression 26

5 Configuration and Setup **27**
5.1 Setup of Hadoop Cluster 27

6 Performance Evaluation **29**

7 Conclusion and Future Scope **33**

8 Screenshots **34**

9 Appendix **40**
9.1 Classes Used in the algorithm 40
9.2 Execution Procedure 41

10 References **42**

List of Abbreviations

CPU	Central Processing Unit
HDFS	Hadoop Distributed File System
NN	Namenode
MR	Map Reduce
RAM	Random Access Memory
SHA	Secure Hash Algorihtm
DES	Data Encryption Standard
MB	Mega Bytes
AES	Advanced Encryption Standard

List of Figures

1.1 General Map Reduce Framework 8
1.2 Hadoop MR Architecture 10

4.1 Overall Proposed Approach 18
4.2 Key Generation Process 19
4.3 Key Generation Algorithm 21
4.4 Map and Reduce Phases 22
4.5 XTS Mode for encrypting a single block of 128 bits 24
4.6 XTS Operation for 128 bits 24
4.7 XTS Operation for blocks greater than 128 bits 25

6.1 Encryption Time for various modes 30
6.2 Compression before Encryption on text data 31
6.3 Compression before Encryption on images 32

8.1 Web Service Login Page 34
8.2 Starting all Hadoop services 35
8.3 File Upload Page 35
8.4 Contents of the user directory after uploading 36
8.5 Encrypted output within the user's directory 37
8.6 Output view in terminal 37
8.7 Browser view of compressed encrypted output 38
8.8 Webpage for retrieving stored files 39
8.9 Decrypted plaintext file 39

Chapter 1

Introduction

1.1 Cloud Storage as a Service

By Moore's law, hardware price per Gigabyte is dropping every day.If too much storage equipments are deployed without full utilization the equipment will be wasted.This has led to the emergence of data centers which provide dedicated storage space to its users.Since such facilities are often shared between several users, the need for special measures to ensure client data integrity arises.

Cloud Storage services (SaaS) have a wide variety of applications.

- Video Surveillance - To store outdated video clips
- Huge data store - ERP, Industry and Consumer statistics
- Backup and archiving - Server/Desktop offsite backup
- Content Distribution - Static content to save bandwidth
- File sharing

1.2 Map Reduce

MapReduce (MR) is a framework for processing huge datasets on certain kinds of distributable problems using a large number of computers (nodes), collectively referred to as a cluster. Computational processing can occur on data stored either in a file system (structured) or within a database (structured). Here we consider the former.

MR was originally proposed and used by Google engineers to process the large amount of data they must analyze on a daily basis. The input data for MR consists of a list of key/value pairs. Map tasks accept the incoming pairs, and map them into intermediate key/value pairs. Each group of intermediate data with the same key is then passed to a specific set of reducers, each of which performs computations on the data and reduce it to one single key/values pair. The sorted output of the reducers is the final result of the MR process. To illustrate MR, we consider an example MR process which counts the frequency of words in a file.

Map tasks accept every single word from the file, and make keys for them. As the task is to count the frequency of all words, a typical approach would be to use word as key. So, for the word "hello", a Map task will generate a key/value pair of hello/5 (if 5 times hello occurs in the file). Afterwards, the key/value pairs with the same key are grouped and sent to reducers. A reducer, which receives a list of values with the same key, can simply count the size of this list, and keep the key in its output. If a Reduce task receives a list with key hello it gathers all such keys and combines them into one key/value pair.
In general, Map Reduce is as follows:

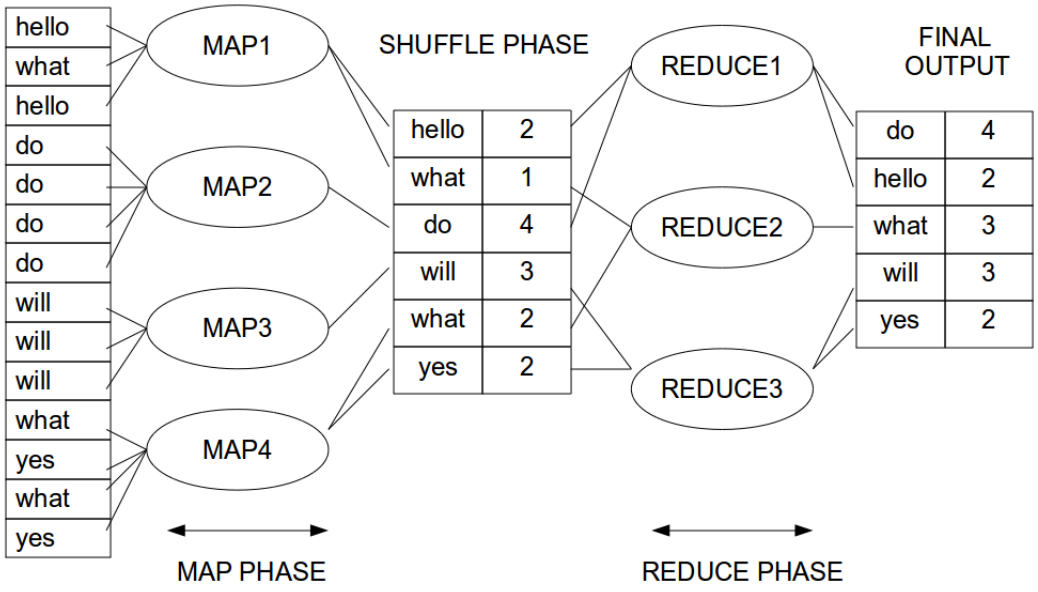

Figure 1.1: General Map Reduce Framework

Map: The master node takes the input, chops it up into smaller sub-problems (small data splits) and distributes those to worker nodes. A worker node may do this again in turn, leading to a multi-level tree structure but here, we restrict with one level. The worker node processes that smaller problem, and passes the answer back to its master node.

Reduce: The master node then takes the answers to all the sub-problems and combines them in a way to get the output - the answer to the problem it was originally trying to solve.

The optional intermediate phase between Map and Reduce is called as Shuffle. In this phase the outputs from the map phase are combined and sorted before being split into reduce splits and given to the reduce tasks.

The advantage of Map-Reduce is that it allows for distributed processing of the map and reduction operations. Provided each mapping operation is independent of the other, all maps can be performed in parallel - though in practice it is limited by the data source and/or the number of CPUs near that data. Similarly, a set of 'reducers' can perform the reduction phase - all that is required is , all outputs of the map operation which share the same key are presented to the same reducer, at the same time. Map-Reduce can be applied to significantly larger datasets than the "commodity" servers can handle - a large server farm can use Map-Reduce to sort peta bytes of data in only a few hours. The parallelism also offers some possibility of recovering from partial failure of servers or storage during the operation: if one map task or reduce task fails, the work can be rescheduled, assuming the input data is still available.

1.2.1 Hadoop's MapReduce

Hadoop's Map Reduce Framework is an open source Project, contributed by Yahoo. It has 2 main components namely the MR engine and the file system.The Map Reduce engine consists of one Job Tracker, to which client applications submit MR jobs. The Job Tracker in turn pushes work out to available TT (Task Tracker) nodes in the cluster.The other main component namely Hadoop Distributed File System (HDFS) will be the main focus of this project.

The HDFS is a distributed, scalable, and portable filesystem written in Java for the Hadoop framework. Each node in an Hadoop instance typically has a single datanode. Each datanode serves up blocks of data over the network us-

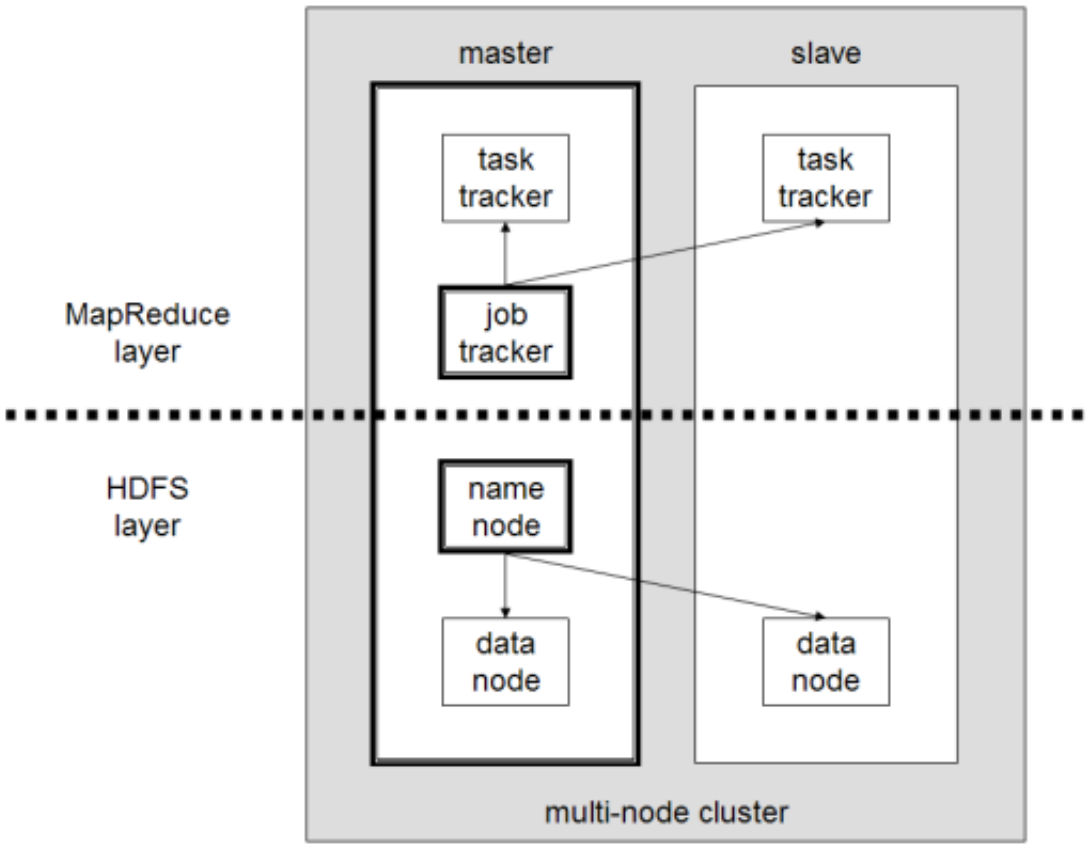

Figure 1.2: Hadoop MR Architecture

ing a block protocol specific to HDFS. The filesystem uses the TCP/IP layer for communication; clients use RPC to communicate between each other. The HDFS stores large files across multiple machines. It achieves reliability by replicating the data across multiple hosts, and hence does not require RAID storage on hosts.Apart from creating multiple replicas of data blocks, HDFS also distributes them on compute nodes throughout a cluster to enable reliable, extremely rapid computations.

Though it is based on UNIX filesystem, HDFS is not fully POSIX compliant.As of version 0.20, despite providing operations to change owner,group and permissions, the authentication mechanism is highly vulnerable.Further,the filesystem requires one unique server, the *name node*. This is a single point of failure for an HDFS installation. If the name node goes down, the filesystem is off-line. When it comes back up, the name node must replay all outstanding

operations. This replay process can be time-consuming.

1.2.2 Fault Tolerance

Since the MapReduce library is designed to help process very large amounts of data using hundreds or thousands of machines, the library must tolerate machine failure gracefully.

1.2.3 Worker Failure

The master pings every worker periodically. If no response is received from a worker in a certain amount of time, the master marks the worker as failed. Any map tasks completed by the worker are reset back to their initial idle state, and therefore become eligible for scheduling on other workers. Similarly, any map task or reduce task in progress on a failed worker is also reset to idle and becomes eligible for rescheduling.

Completed map tasks are re-executed on a failure because their output is stored on the local disk(s) of the failed machine and is therefore inaccessible. Completed reduce tasks do not need to be re-executed since their output is stored in a global file system.When a map task is executed first by worker A and then later executed by worker B (because A failed), all workers executing reduce tasks are notified of the re-execution. Any reduce task that has not already read the data from worker A will read the data from worker B.

1.2.4 Master Failure

It is easy to make the master write periodic checkpoints of the master data structures described above. If the master task dies, a new copy can be started from the last checkpointed state. However, given that there is only a single master, its failure is unlikely; therefore our current implementation aborts the MapReduce computation if the master fails. Clients can check for this condition and retry the MapReduce operation if they desire.

1.3 Need for Security in Clouds

A cloud infrastructure used remotely to store data is our main focus. The other model of Cloud computing is to provide Software as a Service i.e applications that can be run over the Internet remotely.

In either of these cases,security has been identified as a vulnerable point. In the data storage model,users need to be assured of the integrity and confidentiality of the data at rest. This is because of the fact that the infrastructure is generally shared by all users. For instance in a Hadoop-deployed cloud, the HDFS user directory may be common to all users. Unless proper controls are explicitly introduced, user data is vulnerable to malicious access. This gives rise to the need for proper access control mechanisms to data,storage confidentiality and resistance to data modification.Further,since cloud data is stored remotely, there also exist the traditional problems of network security. This problem is magnified by the fact that such data is usually enormous in size.

Major concerns related to huge data are Security, Data Reduction Techniques, Effective Storage and Data Archiving. Of these, security concerns have become quite prevalent, and there is a critical need to secure all data at rest. Despite the fact that many cloud providers use varied mechanisms to secure the cloud, failure to accomplish any one secure mechanism ultimately leads to insecure cloud environment.

Chapter 2

Problem Definition

2.1 Existing Approaches

Security issues for stationary data have been extensively studied for many years. Many of these studies also focus on the current trend of remote data storage. However due to the large data size, most algorithm focus on data processing integrity of the map reduce jobs rather than confidentiality of the data at rest.The time complexity of the encryption algorithm is one of the fundamental factors. In addition to time complexity, effective storage space utilization is another objective of the algorithm.

Algorithms to preserve the processing and service integrity are most favored by researchers. In [1], a prototype of Secure MR is implemented which uses a decentralized replication-based integrity verification scheme for ensuring the integrity of MapReduce in open systems.While this approach prevents modification of results of the Map Reduce jobs,the data stored in HDFS is still vulnerable.

The other approach to security by Yahoo in the latest versions of Hadoop is to link its authentication protocol to Kerberos[10].This approaches user authentication as RPC using SASL mechanism.However, the token based authentication suffers from various flaws as pointed out in [8]. The Yahoo Security design also includes modules for HTTP authentication and ACLs. But disk encryption and data confidentiality is not handled.

MR [4] is designed for building large commodity cluster, which consists of thousands of nodes by using commodity hardware. Hadoop [12], a popular open source implementation of MR framework, developed primarily by Yahoo,

is already used for processing hundreds of terabytes of data on at least 10,000 cores. In this environment, many people share the same cluster for different purpose. Hence MR is widely used in several areas including data analysis [13], warehousing [16] and scientific research [15].

Lori Kaufmann's study of existing cloud architecture and virtual environments outlines the concerns due to public nature of cloud[3].The security policy is alleviated to both service providers and users. This is also studied by other researchers. For example in [11] and [6] , study is made on securing large scale data across distributed across clusters and methods to secure the Map/Reduce tasks.These papers give a view of possible attacks on the worker nodes and tampering of intermediate results, while proposing solutions for the same.

Schneier and Whiting's extensive study [5] explores the implementation of various AES finalists such as Rijndael, Twofish, MARS, RC6. The studies indicate that Rijndael's AES is more suited for applications involving large scale data. The tests are performed on a variety of CPUs. Among the various modes of AES encryption ,[2] refers to XTS mode as most suitable for parallelization. Also [2] is a document to standardize XTS mode by NIST.

Key generation and management is another issue in storage security. Since the design is made on a basis authenticating various users, the symmetric key can be generated as a function of the passwords. Such a key must avoid statistical redundancies of the user password. The mechanism to generate and test for strong keys has been studied extensively in [9].

The various approaches to security and storage of large scale data are studied. In IBM's industrial research [17], it has been proposed to leverage Hadoop in cloud and use its DFS for large scale data storage.

One particularly difficult challenge, that the storage service has to deal with, is to sustain a high I/O throughput in spite of heavy access concurrency to massive data. Massively parallel data transfers need to be performed, which invariably lead to a high bandwidth utilization. With the emergence of cloud computing, data intensive applications become attractive for those who do not have the resources to maintain expensive large scale distributed infrastructures to run such applications. So, minimizing the storage space and bandwidth utilization is highly relevant, as these resources are paid for according to the consumption. Research [18] evaluates the trade-off resulting from applying data compression to conserve space and bandwidth at the cost of slight overhead.

2.2 Problem Statement

The problem is to secure the HDFS by storing only encrypted copies of all user files. The encryption of files is to be done with the control of its owner. Each file must be encrypted in a way unique to the user. Further the keys used for each user must be protected from attacks. The entire process must be done in a time-efficient way while not utilizing excessive resources.

2.3 Proposed Solution

The submitted file is first split into blocks which will be stored in a distributed manner across the nodes. The existing approach in Hadoop is to assign block ids and store each in block in any of the nodes that are designated as data nodes. In order to exploit the inherent parallelism of the distributed architecture and also to reduce the overall execution time, we propose the following methodology. Each block is encrypted in parallel using the Hadoop MR engine.When the file is submitted, a Hadoop process is immediately started in the background and blocks are encrypted as they arrive.The encrypted data blocks are then stored in the HDFS.

The encryption algorithm and mode chosen is viable to parallelism. To avoid attacks where an attacker duplicates the user's data and finds the keys by brute-force, a unique key is generated per user. The key generation and management is part of the security design. The key generation is a function of the username or some other identifier that determines the user uniquely.

Since the size of encrypted data is liable to be much larger, data compression algorithms are applied to utilize available resources efficiently. The problem does not warrant any new technique to show large space savings and hence the existing compression techniques are studied and suitably applied on the user data.

Chapter 3

System Requirements

3.1 Hardware Specification

Master [1]
ProLiant DL3805
4 GB RAM
Smart Array 8 * HP 146 GB HDD

Slaves [32]
ProLiant DL320G3
1 GB RAM
HP 80 GB HDD
Intel Xeon Processor

3.2 Software Specification

Operating System : Debian Lenny GNU/Linux 5.0
Programming Languages : JDK 1.6+ version , PHP 5.3
Framework Used : Hadoop 0.20.2

Chapter 4

Key Management and Encryption

The file submitted to the cluster is split into several blocks each of which is an input to a map reduce tasks and scheduled for execution. The encrypted results are stored into HDFS. The master node has the web server running in it. To encrypt data, a secure key is generated . The modules of the proposed approach are explained in Figure 4.1

4.1 Security of Keys

The first step in the proposed technique is to generate a unique and independent key.The 128 bit symmetric key must have the following properties.

- The key should be kept secret and must be completely random. The entropy must be high. The number of 0s and 1s in the binary form of the key must be nearly equal.
- It should not follow a particular pattern. The probability of any character occurring in the key string must be the same.
- The key has to be generated such that the user has control over the data. This is made possible by designing the key generation algorithm as a function of the username and password.
- The key should be strong enough so that it is not vulnerable to attacks. The key must not be stored as such in any database or filesystem.

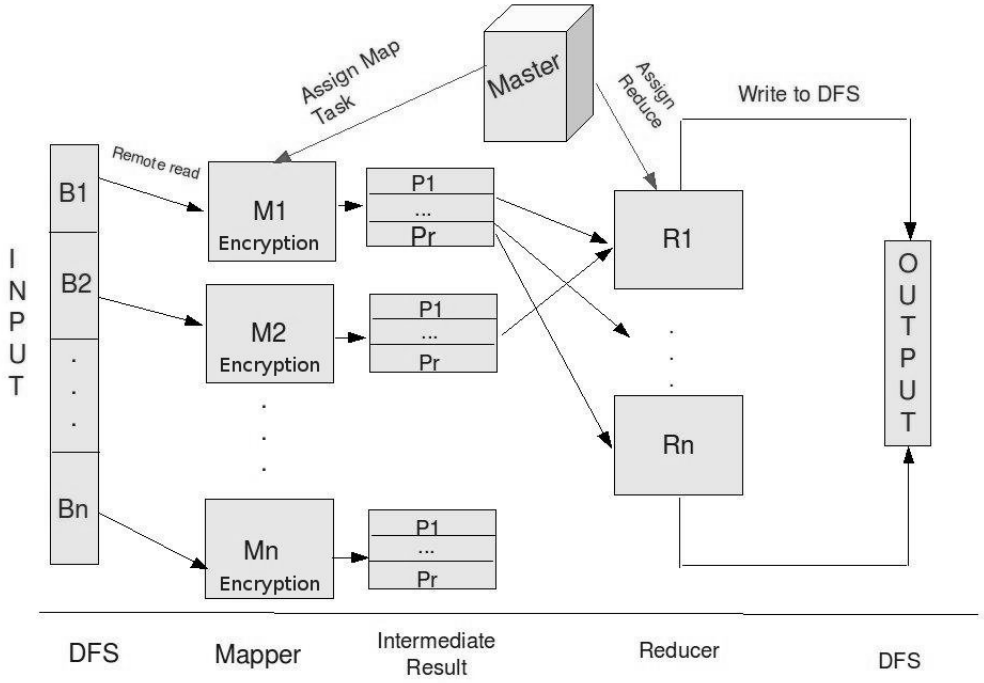

Figure 4.1: Overall Proposed Approach

- Even though the key is a function of the username and password, an attacker knowing the user's login details or any other details of the user, should not be able to predict the key.

4.2 Secure Key Generation

Based on the requirements outlined in Section 4.1, a suitable 128 bit key is generated using the following methodology.

4.2.1 Algorithm for Key Generation

Once the user is authenticated and logs into the web service for file upload, a password is required for each file uploaded. This password is different from the password used for authentication. The password for each file is hashed using SHA-1 algorithm. This hash value alone is stored in the database, with which

a hacker cannot predict the key or password.

The hash is now encrypted using DES algorithm with user name as the key. This is to ensure a unique key per user even if two different users provide the same file password. The hash value generated for the file password will be the same, if the passwords are same. The hash value is encrypted with the user name as key. So, the encrypted hash key generated will be unique. This level of encryption also has the additional benefit of avoiding brute force attacks trying out commonly used passwords.This process can be viewed in Fig 4.2.

The encrypted hash value will be used as the key. The encryption program will use this key during the encryption process. After the encryption process, the key and the input file will be deleted.During decryption, the the user provides the same file password which was used for that file during file upload. The same process happens, hashing and encrypting the hash value with user name as key. So the same encrypted hash value is generated as it was generated during the file upload, which is used as the key for decryption process.

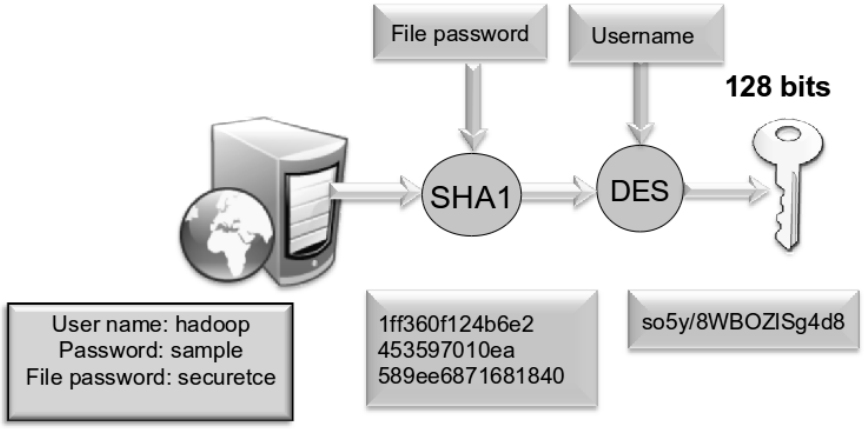

Figure 4.2: Key Generation Process

4.2.2 Key Management

Key Management deals with the storage, transfer, protection and reuse of keys after generation.

There are two key design issues in the above algorithm that will help in effective management of keys. The first design factor is not to store keys in any form

in any database or file. Instead, when keys are needed again the algorithm is applied to regenerate the keys.The key is only a 128-bit value and its generation is part of the PHP web page itself rather than a separate external code.This makes the time for key generation in the order of milliseconds and hence it is no major hindrance to the user to regenerate the key for decryption rather than read it from a database.Thus protection is guaranteed at a low cost.

The second design issue is to have separate login and file upload passwords. If a single password were to be used, users who forget this password may lose all their files. A password-per-file is to avoid such a situation. The user may also opt for convenience of a single password instead of the security offered by multiple passwords. An additional level of protection may be implemented in the form of Security Challenge Questions if the file password is lost by the user. As mentioned in Section 4.2.1, the file password's SHA is stored in the database. If the user clears the security challenge, the password cannot be retrieved but the key can be regenerated (refer Fig 4.2) and hence the file is retrieved. Thus the probability of user losing all the files is minimized.

4.3 Encryption

Using the key generated by the method outlined above, the data blocks are encrypted in parallel by the mapper nodes. Encryption algorithms are extensively used for providing authentication,privacy,confidentiality and non-repudiation. Several standards like PCI,DSS,HIPAA require critical data to be stored in encrypted form. For storage encryption, the symmetric AES algorithm has been selected.

Towards the end of the 20^{th} century many back-doors and flaws were found in the existing symmetric algorithms like DES which is vulnerable to brute-force attacks because of its relatively small 56 bit key size. To meet better security standard, US Government agency NIST selected Rijndael's Algorithm as Advanced Encryption Standard(AES) and it has now become a industry standard. AES is typically designed to accept three different key sizes like 128,192 or 256 bits. The algorithm is found to exhibit good avalanche characteristics i.e. even a change of one bit in the key or plaintext is reflected as a completely different ciphertext. It is capable of encrypting bulk data on top-end 32 bit and 64 bit CPU's. It is efficient in encrypting all sorts of data deployed in cloud from text to audio and video. The performance of AES algorithm vary dramatically on various CPUs based on key size and it can be improved remarkably when it is

```
begin
Input: user name, password and file password
SHA1_hash -> H(file password)
Key -> E(SHA1_hash, user name)
    if(Key.length() < 16)
        while(Key.length() <= 16)
            Key = Key + SHA1(password);
    else if (Key.length() > 16)

        Remove the trailing characters;
    else
        save the 128 bit key in a file;
end
```

Figure 4.3: Key Generation Algorithm

parallelized. These factors make AES a prefect fit as the encryption algorithm for large-scale cloud data.

4.3.1 Map Phase

Block <I_r , object$_r$> is given to mapper M_r.The mapper will generate the corresponding encrypted output I'$_r$ and send it to the reducer R_r
Let W = < I_0 ,object$_0$>, <I_1,object$_1$>, <I_2,object$_2$>.....<I_n,object$_n$ > then we can say that
I'$_r$= M_r (<blockid,Enc-comp$_{object}$>)

4.3.2 Reduce Phase

The collected outputs from various mappers are written to the disk in the sequential order(I'$_1$,I'$_2$,.....I'$_n$). The map and reduce phases are explained in the figure below.

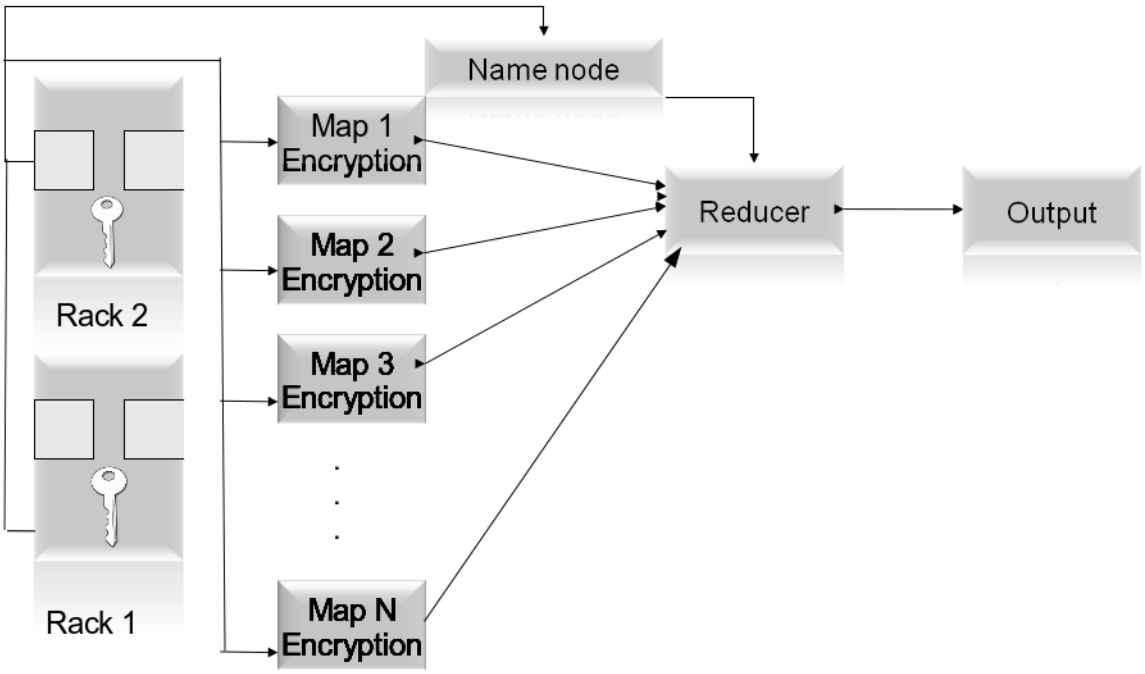

Figure 4.4: Map and Reduce Phases

Since the map phase already generates the encrypted outputs, it is also possible for the reduce phase to be null. In such a scenario, the encrypted output would comprise several files labeled part-m-00*. Encryption without reduce leads to better performance while generating multiple output files instead of a single concatenated output.The results are explored again in full detail in Chapter 6.

4.3.3 Modes of Encryption

AES block cipher by itself allows encryption only of a single data block of the cipher's block length. When targeting a variable-length message, modes of operation enable the repeated and secure use of a block cipher under a single key. Most of the available modes like CTS, CFB, OFB etc. are chained-block modes or feedback modes (i.e) the output of one block is dependent on the previous block's output. For parallelization, each block must be independently processed so that several blocks can be encrypted at once.The 2 possible modes of encryption lending themselves to parallelism are ECB and XTS.

Electronic Code Book (ECB) Mode

ECB supports the kind of parallel encryption in which the plaintext consists of a sequence of blocks like b_1,b_2,.....b_n that is converted into corresponding ciphertext blocks such as c_1,c_2,.....c_n through the same key that acts on all the blocks. As a result if two blocks are same, the same cipher text results. This enables an attacker to figure out all instances of a plaintext if that plaintext-cipher text pair is known and the cipher text is repeating. An attack based on the frequency analysis of the blocks is also possible. Frequently repeating cipher text blocks mean frequently repeating plain text blocks.

XEX-TCB-CTS (XTS) mode

The other widely used parallel mode is XTS which is deemed to be more secure. It has been incorporated in IEEE 1619-2007 standard. The key factor for XTS-AES is that it consists of a encryption key as well as a tweak key that is used to incorporate the logical position of the data block into the encryption. The XTS-AES mode allows parallelization and pipelining in cipher implementations. It enables the encryption of last incomplete block of data while other modes are not having this facility. Since a tweak key is used, repetitive text provides different cipher because of change in position. The operation of XTS mode is illustrated below.

XTS-AES-blockEnc of a single 128-bit block:

With reference to Fig 4.5 , the XTS-AES encryption procedure for a single 128-bit block is modeled with the following equation.

C = XTS-AES-blockEnc(Key, P, i, j)

where
Key is the XTS-AES key
P is a block of 128 bits
i is the value of the 128-bit tweak
j is the sequential number of the 128-bit block inside the data unit
C is the block of 128 bits of ciphertext resulting from the operation

The first operation in Fig 4.6 is modular multiplication with the primitive element over GF(2^{128}). This is followed by usual AES and XOR operations.

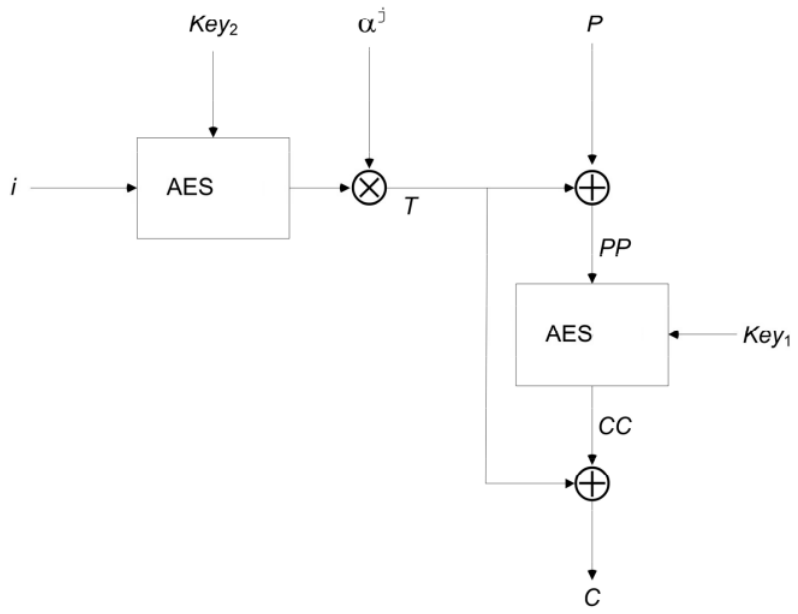

Figure 4.5: XTS Mode for encrypting a single block of 128 bits

$$T \leftarrow \text{AES-enc}(\text{Key2}, i) \otimes \alpha j$$
$$PP \leftarrow P \oplus T$$
$$CC \leftarrow \text{AES-enc}(\text{Key1}, PP)$$
$$C \leftarrow CC \oplus T$$

Figure 4.6: XTS Operation for 128 bits

a dedicated Hadoop user account for running Hadoop. While thats not required it is recommended because it helps to separate the Hadoop installation from other software applications and user accounts running on the same machine

XTS-AES encryption of a data unit:

The XTS-AES encryption procedure for a data unit of plaintext of more than 128 bits is modeled with the following Equation.

$$\boldsymbol{C = XTS\text{-}AES\text{-}Enc\ (Key,\ P,\ i)}$$

where
Key is the XTS-AES key
P is the plaintext
i is the value of the 128-bit tweak
C is the ciphertext resulting from the operation

The plaintext data unit is first partitioned into m + 1 blocks, as follows:
P = P0 —... —Pm1—Pm.
The first m blocks P0,...,Pm1 are each exactly 128 bits long, and the last block Pm is between 0 and 127 bits long (Pm could be empty, i.e., 0 bits long). The key is parsed as a concatenation of two fields of equal size called Key1 and Key2 such that: Key = Key1 l Key2. The ciphertext C is then computed by the sequence of steps shown in Fig 4.7. The procedure for encrypting the last incomplete block (Pm is not 128 bits) is also incorporated in the XTS operation.

```
for q ← 0 to m-2 d
        Cq ← XTS-AES-blockEnc(Key, Pq, i, q)
b ← bit-size of Pm
if b = 0 then do
        Cm-1 ← XTS-AES-blockEnc(Key, Pm-1, i, m-1)
        Cm ← empty
else do
        CC ← XTS-AES-blockEnc(Key, Pm-1, i, m-1)
        Cm ← first b bits of CC
        CP ← last (128-b) bits of CC
        PP ← Pm | CP
        Cm-1 ← XTS-AES-blockEnc(Key, PP, i, m)
C ← C0|...|Cm-1|Cm
```

Figure 4.7: XTS Operation for blocks greater than 128 bits

4.4 Compression

Compression is the process of exploiting statistical redundancy in the data and represent it using fewer bits.

On encryption, our results show that the size of data has nearly doubled. To utilize the available resources efficiently, a known compression algorithm such as LZO, GZip or BZip is applied on the input file. The above methodology is modified only to compress each block in the mapper phase before encryption. HDFS also supports record level and block level compression on input data. Compression ratio in the order of nearly 1:10 for plain text files is achieved as shown in Chapter 6. The compression can either be part of the mapper code before encryption or be done on the final encrypted output. From a security point of view, compression before encryption dissolves the regularities present in the plaintext. Hence it is preferable to be done as part of the Mapper code itself.

Chapter 5

Configuration and Setup

Hadoop is first installed and the various configuration parameters are set. For the Encryption code to be executed, some configuration has to be done in the slave and master nodes. In the master node, a LAMP web server is setup and the required PHP scripts for the web interface are hosted. Also the java code is first placed in the HADOOP_HOME directory. The code is compiled and converted into a **jar** file and has to be run from the **www-data** folder in the project.

Also password less secure shell login (SSH) from the master into all the slave nodes are configured for the data transfer to take place. The shell script files necessary to run the map reduce tasks are also called from web scripts.

5.1 Setup of Hadoop Cluster

The procedure for installing Hadoop is outlined here. The version chosen was hadoop-0.20.2 from the official Apache repositories. First, a dedicated Hadoop user account is setup for running Hadoop. While that's not required, it is recommended because it helps to separate the Hadoop installation from other software applications and user accounts running on the same machine.

Next to enable passwordless SSH for the Hadoop user, the **ssh-keygen** utilty is used to create RSA key pair for the machine. This key is then placed in the *authorized_keys* file of the Hadoop user's home directory in the slaves so that hadoop can authorize itself to all nodes without requiring a password.

Then the Hadoop package is placed in the hadoop user's home, and exported as the environmental variable HADOOP_HOME. Now the various configuration

parameters are set to run the MapReduce engine as well as the HDFS.

First in hadoop-env.sh file, the environmental variable JAVA_HOME is pointed to the correct classpath of the java installation. Now some important properties need to be set in the various hadoop configuration files. First, the core-site properties need to be set. These are hadoop.tmp.dir - a base location for temporary files and fs.default.name which specifies a URI with a unique port whose scheme and authority determine the FileSystem implementation.

Finally, some properties need to be set in the mapred-site and the hdfs-site files which control the corresponding components of Hadoop. These include the host and port that the MapReduce job tracker runs at and the replication factor for HDFS. When these essential properties are set, Hadoop is ready to run.

Finally, to execute the AES code, all Hadoop services need to be started using the HADOOP_HOME/bin/start-all.sh script.

Chapter 6

Performance Evaluation

In this section, a number of experiments are carried out which outlines the effectiveness of the proposed algorithm. The purpose of these experiments is to note how the performance of the proposed algorithm is better from the existing algorithm.All experiments were tested on a 32 node HP Proliant cluster.

In Figure 6.1, we compare the encryption times for AES in the two modes and also study the effect of setting reducer to null.As seen from the data in Fig 6.1 (i) and (ii), the increase in time due to reducer phase is high.This is because the reducer is in this case a sequential process that concatenates together the mapper outputs.So it is preferable to have multiple outputs in favor of time efficiency.

Also the results from Fig 6.1 (iii) clearly show a marginal increase in time for XTS mode over ECB. But this is an acceptable trade-off to avoid attacks outlined in Section 4.3.3 to which ECB is easily vulnerable. Hence from these results, it is clear that AES-XTS is the most secure and efficient mode of operation.

The next set of evaluations is to determine the efficiency and necessity of compression. First consider the results for text data in Fig 6.2. Again there is a trade-off between compression and execution time. Here the choice may be vendor-specific. As a compression ratio of nearly 1:9 is achieved on text data, the small increase in time may be worth on storage space by nearly a factor of 10.

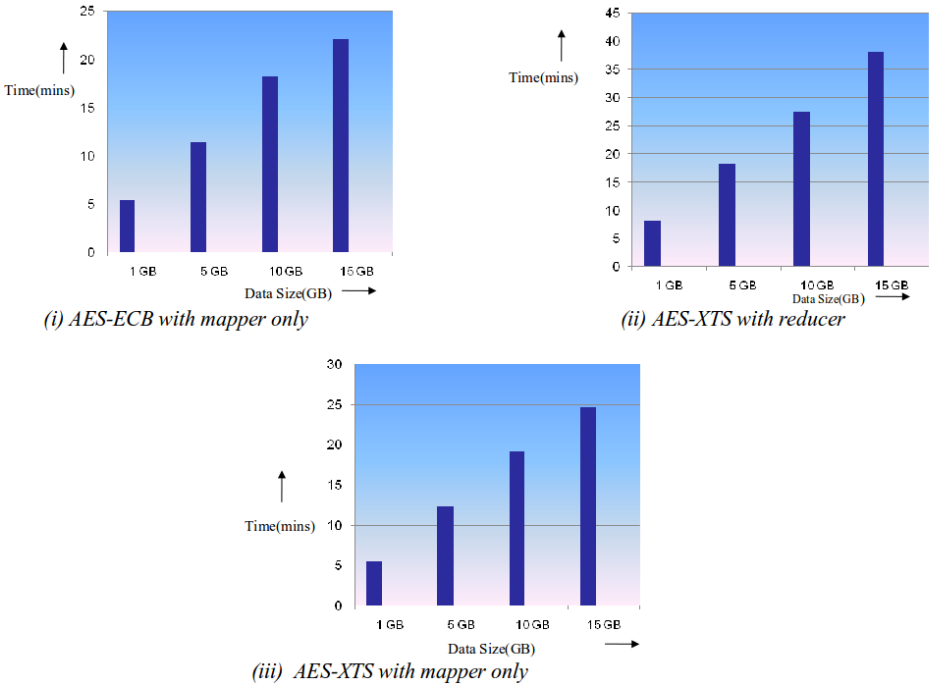

Figure 6.1: Encryption Time for various modes

Next, the efficiency of our Key Generation algorithm is tested. The various requirements of a strong key laid out in Section 4.1 call for randomness in the symbols contained in the key. Since user passwords are generally repetitive and contain dictionary words, the key generation algorithm must remove all redundancies. In the results shown in Table 6.1, we test the generated keys for various types of commonly used passwords. First a very short password with repeated characters is used. Even in this case, the generated key is a random alphanumeric string with no repetition. Similarly results are shown with password consisting only numbers, only alpahabets and a regular alpha numeric password.

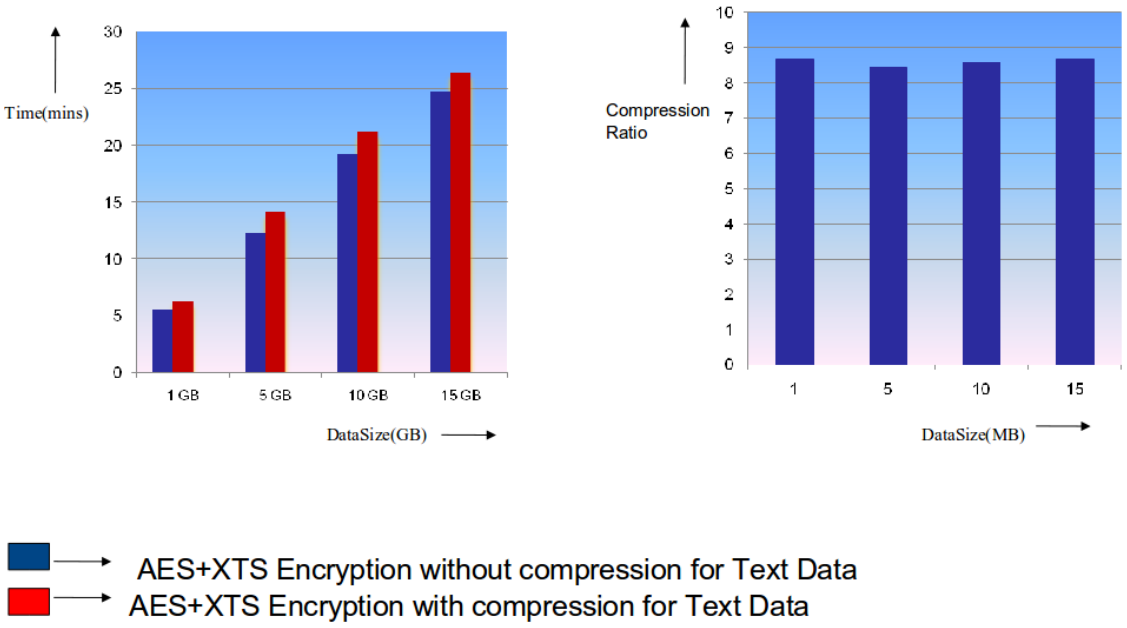

Figure 6.2: Compression before Encryption on text data

Table 6.1: Key Generation Algorithm

Password	Key	Key in Hex
aaa	se307EW2K9oYA	73653330374557324B396F5941
1234	haXBsa0I08mhY	6861584273613049303 86D6859
ksbd	dotPWxh.mmZNU	646F74505778682E6D6D5A4E5
sample	te.tMSAp/bq2Y	74652E744D5341702F62713259
nextpage99	daCBCCaQIIfGw	64614342434361514949664777

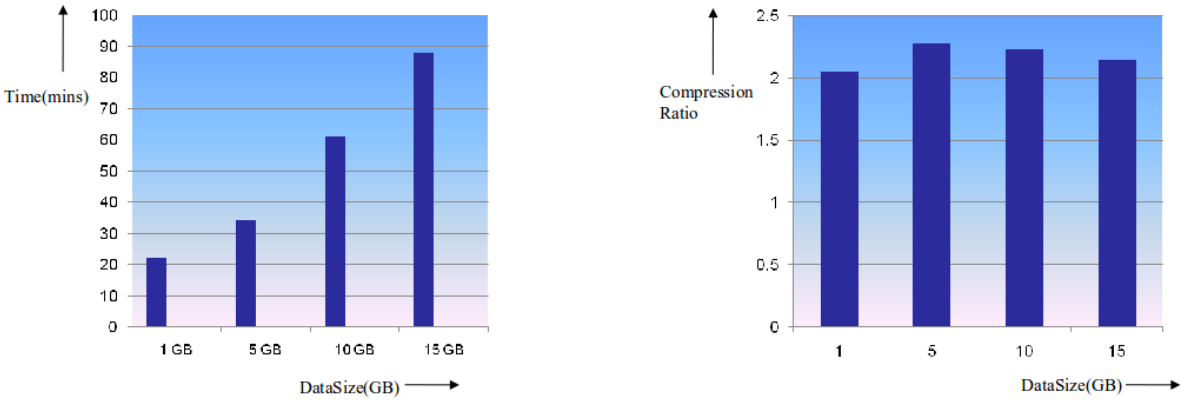

Figure 6.3: Compression before Encryption on images

The above set of results (Fig 6.3) extend the study to image data sets. Due to the bulky nature of images, the time axis shows a massive increase. Also images are not as easily compressible as plaintext but still the compression ratio graph of Fig 6.3 shows a savings on storage for images by a factor of 2 to 2.5.

Chapter 7

Conclusion and Future Scope

The algorithm proposes an effective way to maintain confidentiality of data at rest with focus on HDFS. This can be made scalable to distributed systems in general.Another possible enhancement on the algorithm is to extend it to provide processing integrity allowing users to run and fetch Map Reduce results securely. The encryption can also be done on top of a bucket-based system to incorporate access control features on the DFS. HDFS now provides mountable support through FUSE services like WebDAV. The encryption can also be extended to the mounted HDFS. Finally, the compression technique can be improved to obtain more space efficiency through folder level Deduplication i.e creating symbolic links for files with same content. Techniques to improve image compression ratio like Fuzzy matching is also a possible area of research.

Chapter 8

Screenshots

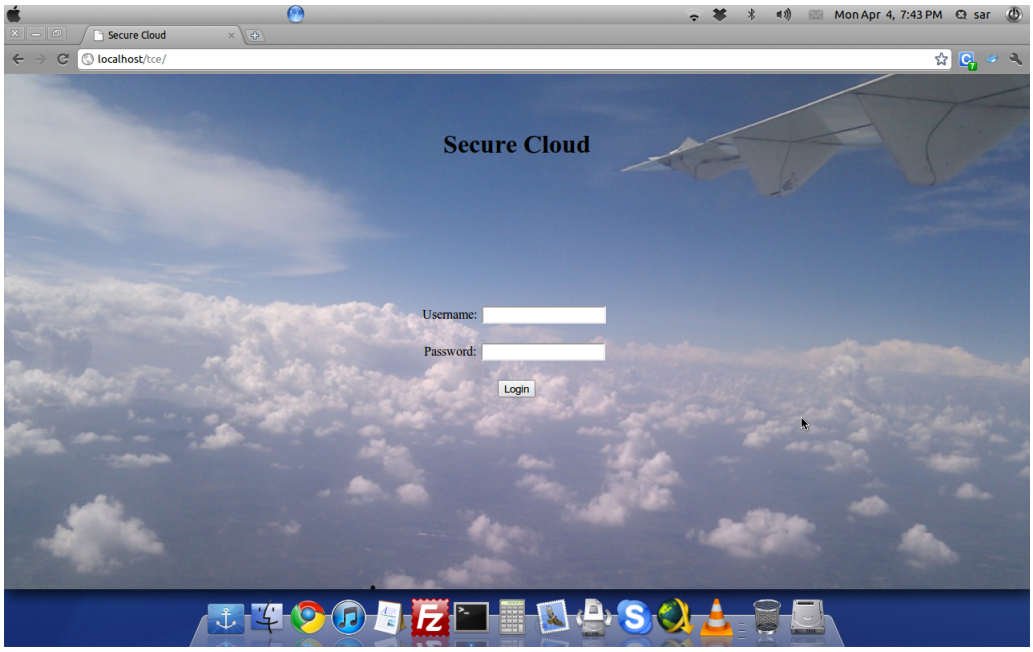

Figure 8.1: Web Service Login Page

Figure 8.2: Starting all Hadoop services

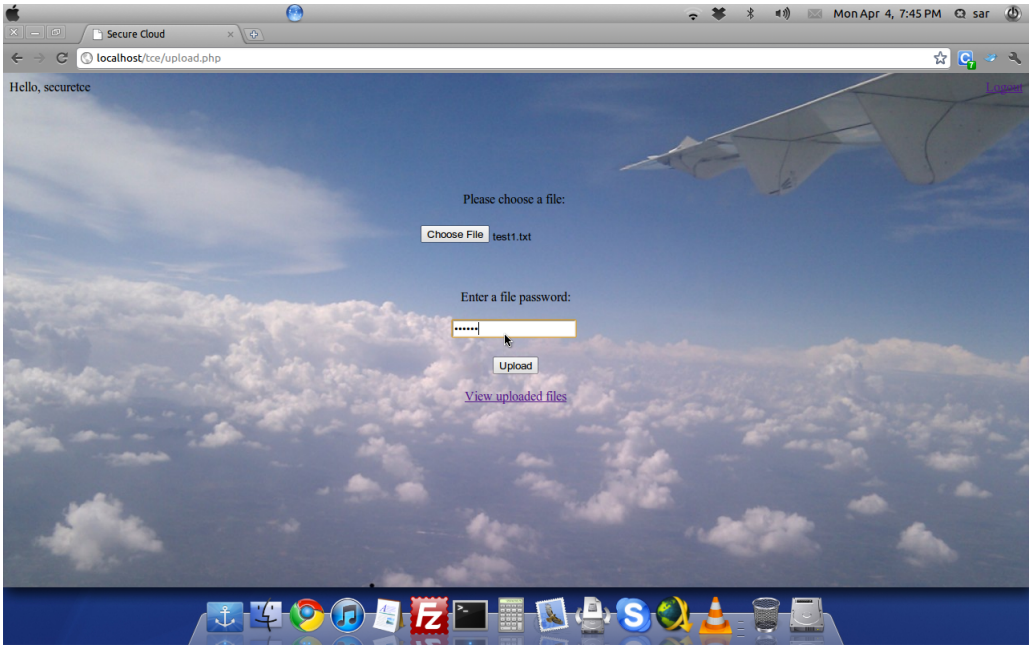

Figure 8.3: File Upload Page

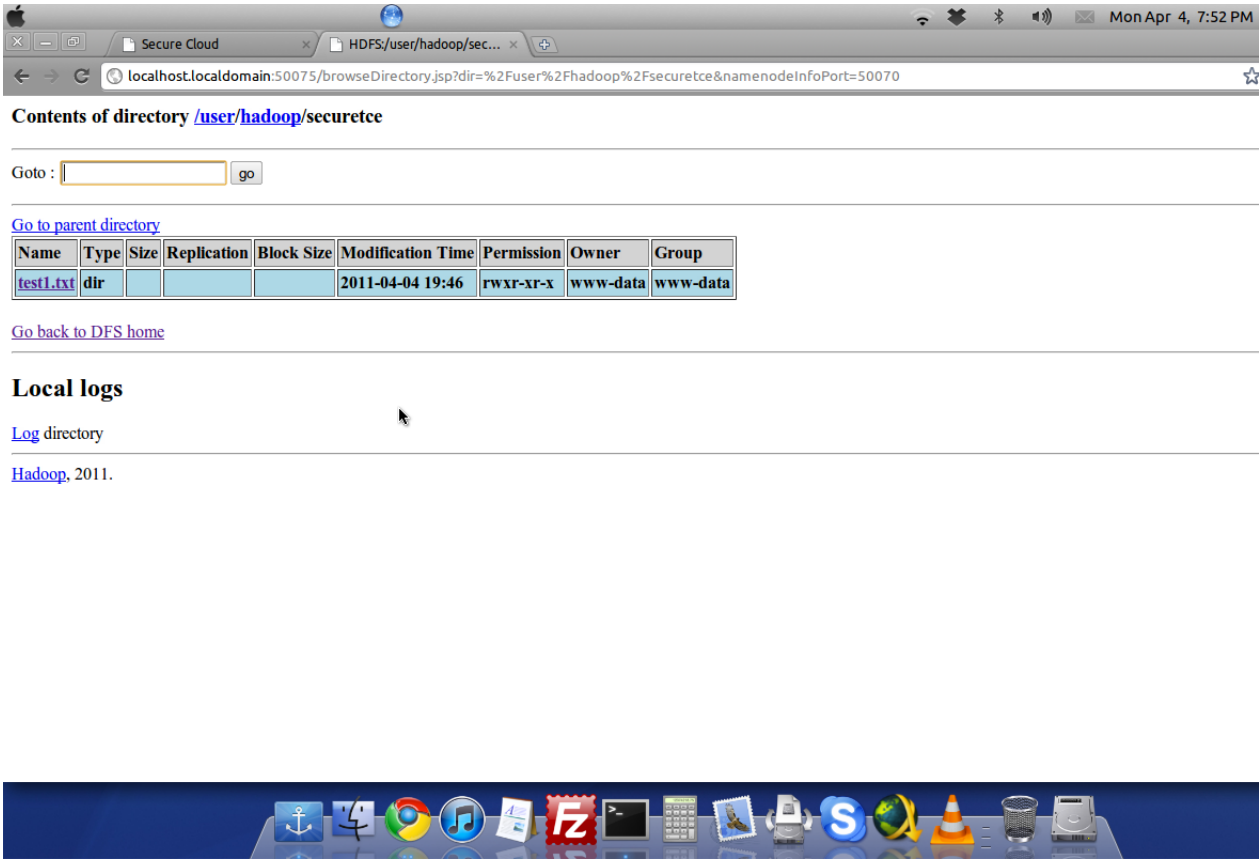

Figure 8.4: Contents of the user directory after uploading

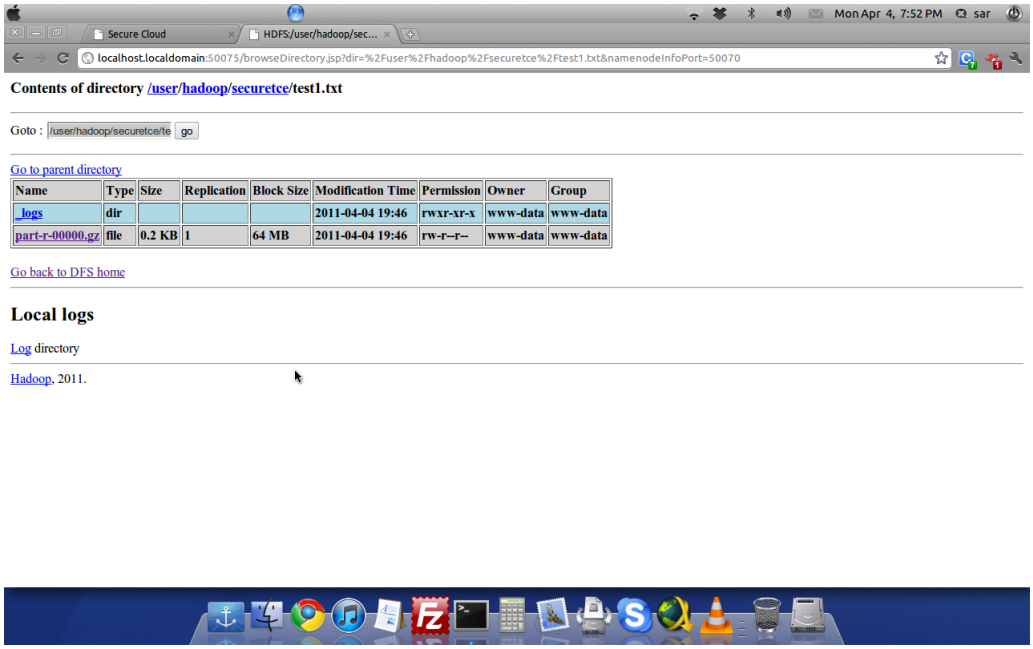

Figure 8.5: Encrypted output within the user's directory

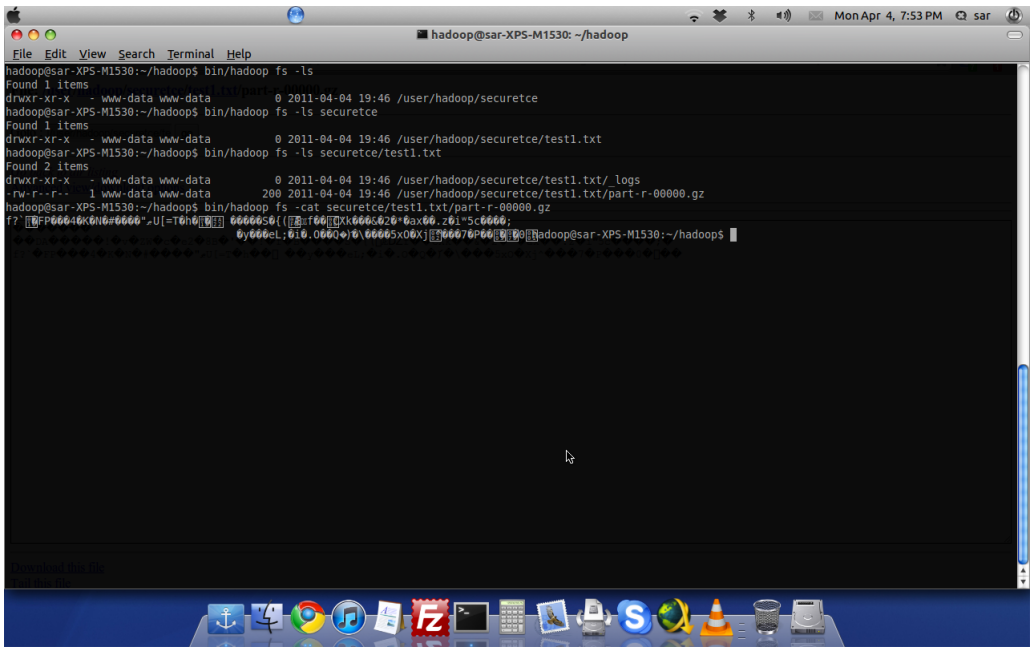

Figure 8.6: Output view in terminal

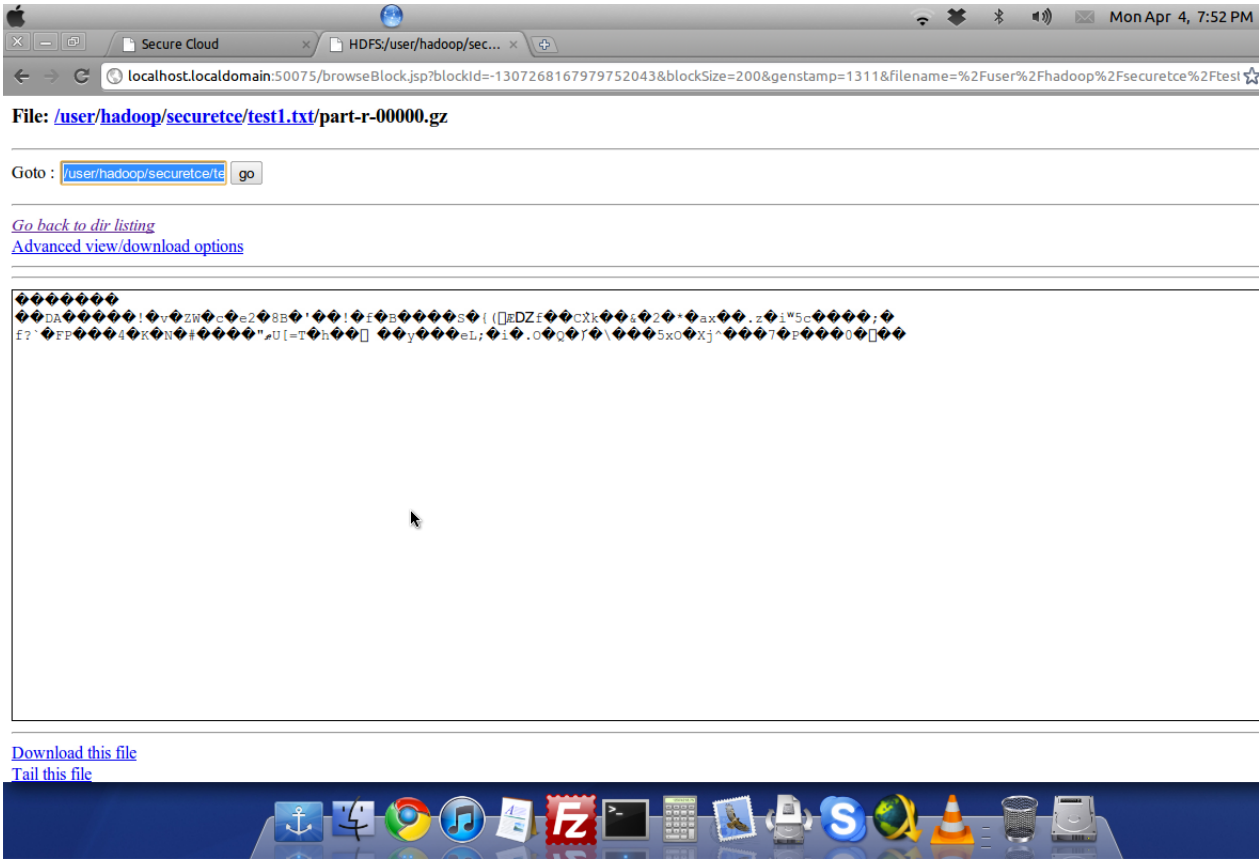

Figure 8.7: Browser view of compressed encrypted output

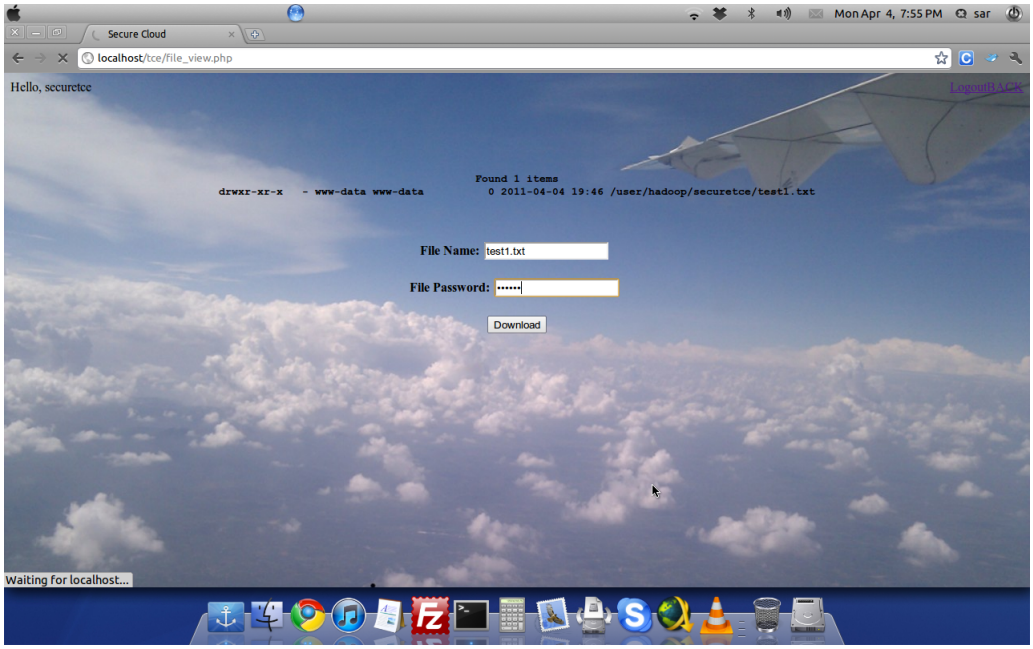

Figure 8.8: Webpage for retrieving stored files

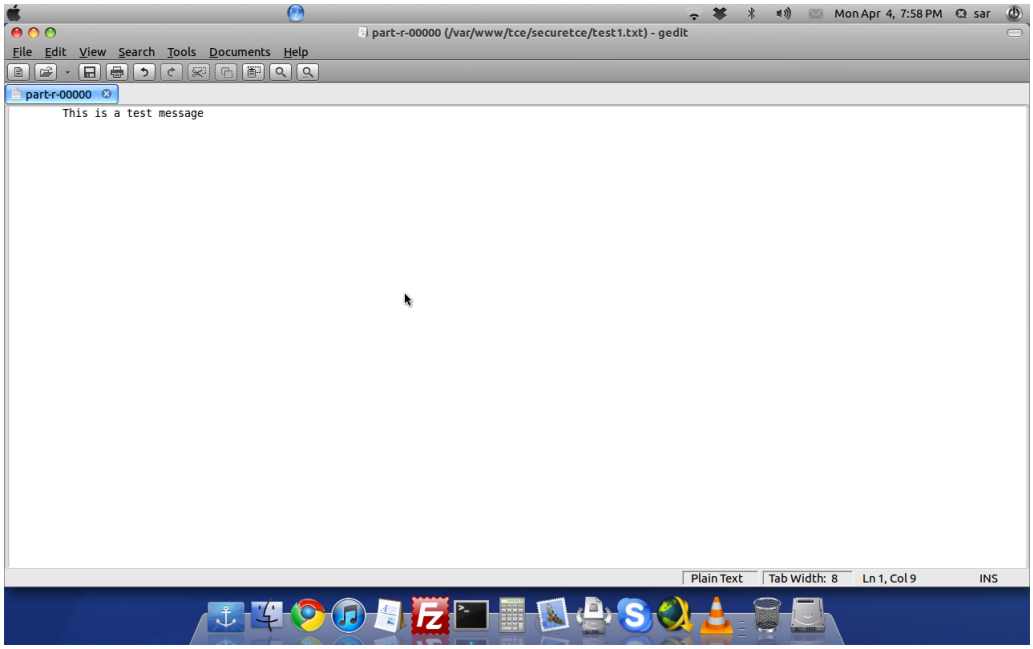

Figure 8.9: Decrypted plaintext file

Chapter 9

Appendix

9.1 Classes Used in the algorithm

```
class MapClass {
int bcount;
int blockid;
String word;
String encrypted;
int map_id;
Cipher enc;
byte[] result;
SecretKeySpec skey;
}

class Reduce {
String reduce_id;
String result;
}

class aesenc {
String username;
String hexbuffer;
String block_id;
FileSystem hdfs;
FSDataInputStream file_in;
}

class aesdec {
```

String username;
String data;
String block_id;
FileSystem local_fs;
FSDataInputStream file_in;
}

9.2 Execution Procedure

Hadoop configuration and password less SSH mentioned in Section 5 has to be done, then the AES-Encrypt and AES-Decrypt jar code in the web server directory (www-data) of the project is run using the java interpreter of JDK 1.6+ version which is invoked from the PHP script. The web application is started in the browser using the url http://master-ip/securehadoop where securehadoop is the name of the project folder.Also a MySQL database has to be setup to check the login details.

- The user logs in using a password provided by the service provider.This authentication is verified across the MySQL DB.
- Start hadoop services in the master node by traversing to HADOOP_HOME directory and executing the command bin/start-all.sh
- In the upload page of the browser, any file input is given with an upload password of the user's choice
- After the background map reduce job is over, the results can be checked in the port 50070 as http://master-ip:50070
- The resultant encrypted output is stored as part-r-000*.gz in username/filename directory.
- Results are viewed in terminal using bin/hadoop dfs -cat command.
- In the web page, the "Decrypt" link lists the available files for that user and any file can be decrypted and downloaded by supplying its password.

Chapter 10

References

1. Wei Wei, Juan Du, Ting Yu, Xiaohui Gu, **"SecureMR: A Service Integrity Assurance Framework for MapReduce "**, *IEEE Annual Computer Security Applications Conference*, May 2009.

2. M.Dworkin **"Recommendation for Block Cipher Modes of Operation:The XTS-AES Mode for confidentiality on Storage Devices"**, *NIST Special Publication 800-38E, US National Inst. Of Standards and Tech,2010.*

3. Lori M.Kaufmann, **"Data Security in the world of Cloud Computing"**, *IEEE Security and Privacy* , Vol2, p61-64,2010.

4. Jeffrey Dean and Sanjay Ghemawat , **"MapReduce: Simplified Data Processing on Large Clusters"**, *Communications of the ACM* , Vol.51, No 1, 2008.

5. Bruce Schneier and Doug Whiting, **"A Performance Comparison of the Five AES Finalist"**, *Second AES Candidate Conference*, 2000.

6. Michael Armbrust, Armando Fox et al **"Above the Clouds: Berkeley View of Cloud Computing"**,*Electrical Engineering and Computer Sciences University of Berkeley*, 2009.

7. William Yurcik et al **"Cluster Security as a Unique Problem with Emergent Properties"**, *LCI International Conference*, 2004.

8. Andrew Becherer **"Hadoop Security Design.Just Add Kerberos?"**, *iSec Partners Black Hat Conference*, 2010.

9. Shakir Hussain et al, **"A Password based Key Derivation Algorithm using KBRP"**, *American Journal of Applied Sciences* , 2008.

10. Owen O'Malley **"Integrating Kerberos into Apache Hadoop"**, *Yahoo Kerberos Conference*, 2010.

11. Jason Schlesinger **"Cloud Security in Map/Reduce "**, *DefCon*, July 2009.

12. Andrew Pavlov et al. **"A comparison of approaches to large scale data analysis "**, *ACM Press,SIGMOD*, July 2009.

13. Abouzeid et. al. **"HadoopDB, An architectural hybrid of MapReduce and DBMS technologies for analytical workloads "**, *In proceedings of conf. on Very Large Databases*, July 2009.

14. Brian Warner et al. **"Tahoe - The Least-Authority Filesystem "**, *StorageSS 08,ACM 2008*, March 2008.

15. J. Ekanayake, S. Pallickara, and G. Fox, **"MapReduce for data intensive scientific analysis"**, *eScience*, 2008.

16. Thusoo, A et.al. **"Hive: A warehousing solution over a MapReduce framework"**, *In proceedings of conf. on Very Large Databases*, 2009.

17. Himabindu et al. **"Towards optimizing Hadoop provisioning in the Cloud"**, *IBM Research,Almaden*, 2009.

18. Bogdan Nicolae **"High throughput data-compression for cloud storage"**, *Proceedings of the Third international conference on Data management in grid and peer-to-peer systems*, 2010.

19. http://hadoop.apache.org/core/

20. http://hadoop.apache.org/common/docs/current/api/org/apache/hadoop/security/

21. http://www.gartner.com/technology/research/hype-cycles/