CHAPTER 1

INTRODUCTION

1.1 ABOUT THE PROJECT

This project works on the basis of Network security, and the issues faced when storing information through cloud - based services. The proposed project aims to solve some of the existing security flaws in storing files on the cloud using a cryptographic splitting approach. The study uses an algorithm to perform the split- chunking process. It also has an Encryption and Decryption process to it.

1.2 FILE SELECTION

There are many file formats available for uploading and encryption process. It is very important to establish heterogeneity of process execution with the help of the algorithm. There are many types of files that can by processed by our project as given as follows.

1. Binary Files
2. Executable Files
3. Non- Binary Files

Binary Files:

This is the foremost and simplest file format that every data is converted into. However, encryption and split chunking process works the best in that format.

Executable Files:

Data transferred in an authentic channel is full proof of tampering but the data can be overheard during the transfer.It is very difficult to build a completely secure channel and no such channel exists currently. Although research suggests that through quantum cryptography, which is completely resistant to data tampering and on which data cannot be intercepted can be built.

Non - Binary Files:

Other file formats such as video, audio, documents and spreadsheets can also be encrypted and split. Some Notable file Formats are:

1. mp3
2. avi
3. doc
4. txt
5. xls
6. csv
7. pdf

1.3 CYPTOGRAPHY

Cryptography is defined as constructing techniques which prevent third parties and the public from reading the private messages. Cryptography is synonymous with the term encryption which meant the conversion of a message, which is in a readable state to a form which is not understandable by human.

Cryptography can also be defined as practice and study of techniques for secure communication in the presence of attackers. There are many aspects which need to be taken care of while developing a cryptography technique:

1. Confidentiality
2. Data Integrity
3. Authentication
4. Non-repudiation

Encryption:

The process of changing the form of a message in such a way that only authorized parties have access to the message/information.

Decryption:

The process of converting the encrypted information back to the original form i.e. reversing the encryption process performed on the data.

Algorithm:

An algorithm is a step by step instruction to solve a problem. AN algorithm solves a problem by performing calculations, data processing, automated reasoning, among others. A simple cryptography algorithm reads a message, applies the algorithm with a specified key during encryption and after the receiver receives the message the shared key is used to decrypt the message. There are two types of cryptography:

1. Symmetric key cryptography systems
2. Asymmetric key cryptography systems.

1.4 SYMMETRIC KEY CRYPTOGRAPHY

In symmetric key cryptography the same key is used while encrypting and decrypting the message. There are many problems in key sharing and distributing in this method. It is a best practice to use this method when the same user is going to encrypt and decrypt the message. This method brings in the issue of creating a secure channel for key transfer to the recipient. The secret key used can be a number, word, string of random letters. The secret data could be in threat if an eavesdropper has access to the key. The Plain text is the original message which needs to be transferred. Standard encryption algorithms are applied to the plain text using the key and the Cipher text is the output of the encryption process. The key, cipher text is then transferred through a secure channel to the recipient. The recipient decodes the cipher text using the same algorithm and the same key used for encrypting the message. After successfully decoding the cipher text the receiver can read the original message.

Examples: Blowfish, AES, RC4, DES, RC5, and RC6.

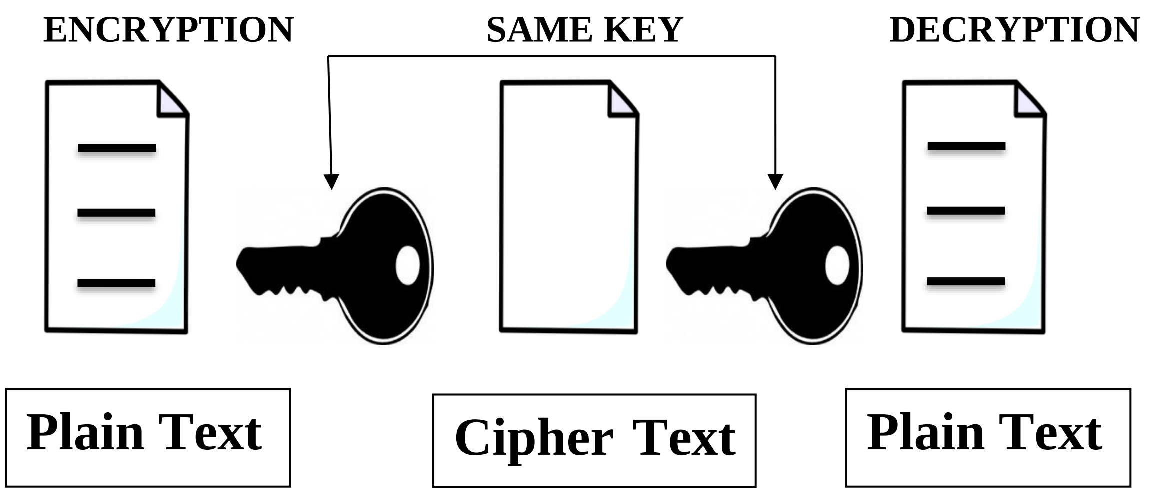

Fig. 1.1 Symmetric Encryption

1.5 ASYMMETRIC KEY CRYPTOGRAPHY

Asymmetric key cryptography is also known as Public Key Cryptography, uses a public key to encrypt a message and a private key to decrypt the message. The private key is known only to the user who is decrypting the message. Keys are exchanged over the Internet or through a local network. This method ensures that attackers cannot misuse the keys. Both the keys are related in some way which helps in increasing the security of the key being hacked. The public key is available to all the users who wants to send a message. The private key is shared only with the user for whom the message is intended for to ensure security. There is no security for the public key as it is available over the internet or on a large network. The message can also be encrypted using the private key and can be decrypted at the sender's end using the public key. Asymmetric cryptography provides better security than Symmetric cryptography as two different keys are used and no sharing of keys takes place.

Examples: El Gamal, RSA, DSA, Elliptic curve techniques, PKCS.

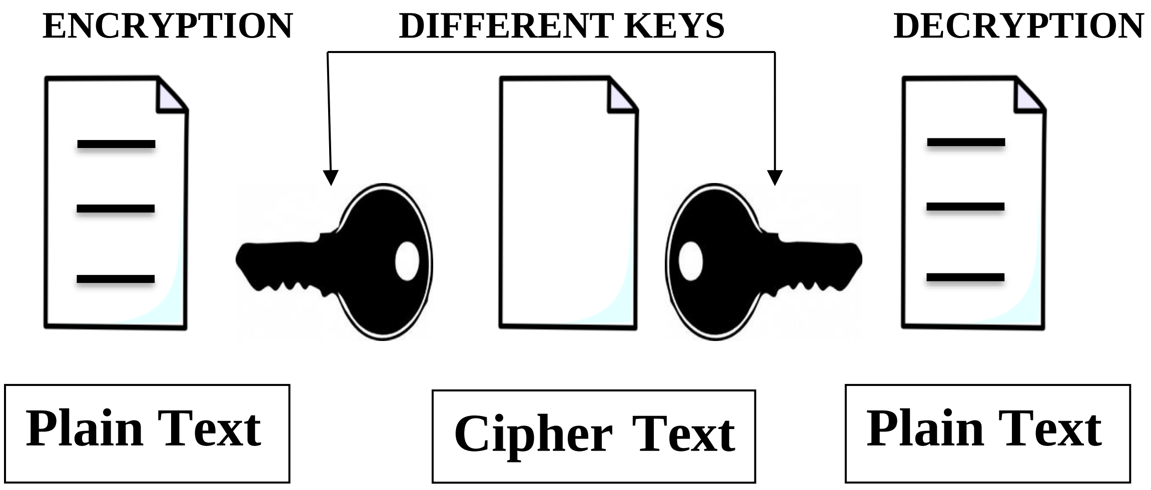

Fig. 1.2 Asymmetric Encryption

1.6 DOMAIN – FILE SECURITY USING CRYPTOGRAPHIC APPROACH (SPLITTING)

Cryptographic splitting, also known as cryptographic bit splitting or cryptographic data splitting, is a technique for securing data over a computer network. The technique involves encrypting data, splitting the encrypted data into smaller data units, distributing those smaller units to different storage locations, and then further encrypting the data at its new location. With this process, the data is protected from security breaches, because even if an intruder is able to retrieve and decrypt one data unit, the information would be useless unless it can be combined with decrypted data units from the other locations.

Cryptographic splitting utilizes a combination of different algorithms to provide the data protection. The whole data is first split into several chunks or pieces, each piece is concurrently encrypted using AES-256 government encryption standard and stored in different locations. Elements required to perform these tasks such as Initialization vector and SALT are generated at random during runtime and stored in the User Key File which is again encrypted with user’s passkey.

All these gives three layers of protection Encryption, Slicing, and Encryption of User Key File generally called as Map file. These layers of security make sure that the file stored in the server cannot be accessed by any Hacker or any malicious intelligent program.

1.7 ARCHITECTURE FOR CRYPTOGRAPHIC FILE SPLITTING

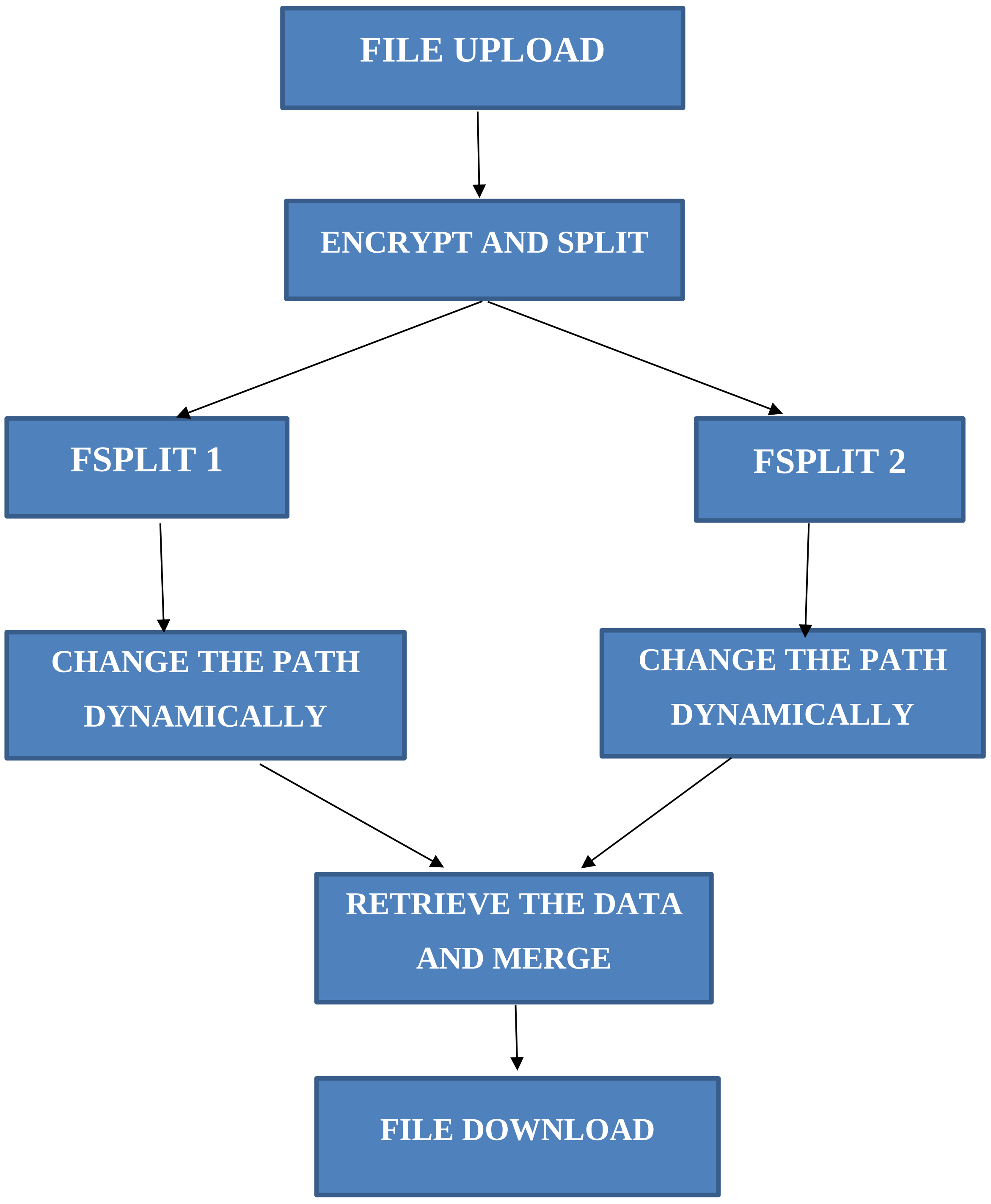
FILE UPLOAD
ENCRYPT AND SPLIT
FSPLIT 1
FSPLIT 2
CHANGE THE PATH DYNAMICALLY
CHANGE THE PATH DYNAMICALLY
RETRIEVE THE DATA AND MERGE
FILE DOWNLOAD

Fig. 1.3 Architecture for Cryptographic Splitting

1.8 PROBLEM STATEMENT

Data security is a big challenge. There are many techniques which help to solve this issue. One such technique is Cryptographic splitting approach. Using this technique one can secure their data to be transmitted in Text, Images, Audio, Video files nearly all kinds of data. The proposed application uses file splitting method by converting the large files into smaller chunks of data. This method makes the intruder unaware of the file segmentation. This process clearly eliminates the capability of obtaining the single file as a whole. This eliminates the risk of file sniffing or packet sniffing. The main issue here is Credibility and Surety. This process guarantees the same for the entire file by preventing data loss. Portability of huge files is also taken into account.

1.9 APPLICATION

The proposed system aims to make use of Image Steganography to hide the secret message to be transmitted to the receiver.

The user has to sign up using the signup page. Using the credentials entered while signing up the user can then login to either embed or decrypt an image file. The embed image page asks for the message/information to be inputted by the user. Then the user chooses an image file in which the message will be embedded. The embedding process is carried out using Least Significant Bit Algorithm. After embedding the message into the chosen image file, the user has the option to save the image file in PNG format on his/her computer or to transfer the embedded image to the recipient by email. The email module uses SMTP protocol for transferring the embedded image.

1.10 TECHNOLOGIES USED

JAVA

It is a Platform Independent. Java is an object-oriented programming language developed initially by James Gosling and colleagues at Sun Microsystems. The language, initially called Oak (named after the oak trees outside Gosling's office), was intended to replace C++, although the feature set better resembles that of Objective C.

1.11 INTRODUCTION TO JAVA

Java has been around since 1991, developed by a small team of Sun Microsystems developers in a project originally called the Green project. The intent of the project was to develop a platform-independent software technology that would be used in the consumer electronics industry. The language that the team created was originally called Oak.

The first implementation of Oak was in a PDA-type device called Star Seven (*7) that consisted of the Oak language, an operating system called GreenOS, a user interface, and hardware. The name *7 was derived from the telephone sequence that was used in the team's office and that was dialed in order to answer any ringing telephone from any other phone in the office.

Around the time the First Person project was floundering in consumer electronics, a new craze was gaining momentum in America; the craze was called "Web surfing." The World Wide Web, a name applied to the Internet's millions of linked HTML documents was suddenly becoming popular for use by the masses. The reason for this was the introduction of a graphical Web browser called Mosaic, developed by NCSA. The browser simplified Web browsing by combining text and graphics into a single interface to eliminate the need for users to learn many confusing UNIX and DOS commands. Navigating around the Web was much easier using Mosaic.

It has only been since 1994 that Oak technology has been applied to the Web. In 1994, two Sun developers created the first version of Hot Java, and then called Web Runner, which is a graphical browser for the Web that exists today. The browser was coded entirely in the Oak language, by this time called Java. Soon after, the Java compiler was rewritten in the Java language from its original C code, thus proving that Java could be used effectively as an application language. Sun introduced Java in May 1995 at the Sun World 95 convention.

Web surfing has become an enormously popular practice among millions of computer users. Until Java, however, the content of information on the Internet has been a bland series of HTML documents. Web users are hungry for applications that are interactive, that users can execute no matter what hardware or software platform they are using, and that travel across heterogeneous networks and do not spread viruses to their computers. Java can create such applications.

a) Working of JAVA

For those who are new to object-oriented programming, the concept of a class will be new to you. Simplistically, a class is the definition for a segment of code that can contain both data (called attributes) and functions (called methods).

When the interpreter executes a class, it looks for a particular method by the name of main, which will sound familiar to C programmers. The main method is passed as a parameter an array of strings (similar to the argv[] of C), and is declared as a static method.

To output text from the program, we execute the println method of System.out, which is java's output stream. UNIX users will appreciate the theory behind such a stream, as it is actually standard output. For those who are instead used to the Wintel platform, it will write the string passed to it to the user's program.

Java consists of two things :

- Programming language
- Platform

b) The JAVA Programming Language

Java is a high-level programming language that is all of the following:

- Simple
- Object-oriented
- Distributed
- Interpreted
- Robust
- Secure
- Architecture-neutral
- Portable
- High-performance
- Multithreaded
- Dynamic

The code and can bring about changes whenever felt necessary. Some of the standard needed to achieve the above-mentioned objectives are as follows:

Java is unusual in that each Java program is both co implied and interpreted. With a compiler, you translate a Java program into an intermediate language called **Java byte codes** – the platform independent codes interpreted by the Java interpreter. With an interpreter, each Java byte code instruction is parsed and run on the computer. Compilation happens just once; interpretation occurs each time the program is executed.

You can think of Java byte codes as the machine code instructions for the Java Virtual Machine (JVM). Every Java interpreter, whether it's a Java development tool or a Web browser that can run Java applets, is an implementation of JVM. That JVM can also be implemented in hardware. Java byte codes help make "write once, run anywhere" possible.

You can compile your Java program into byte codes on any platform that has a Java compiler. The byte codes can then be run on any implementation of the JVM. For example, that same Java program can e run on Windows NT, Solaris and Macintosh

c) The JAVA Platform

A platform is the hardware or software environment in which a program runs. The Java platform differs from most other platforms in that it's a software-only platform that runs on top of other, hardware-based platforms. Most other platforms are described as a combination of hardware and operating system.

The Java platform has two components :

- The Java Virtual Machine (JVM)

- The Java Application Programming Interface (Java API)

You've already been introduced to the JVM. It's the base for the Java platform and is ported onto various hardware-based platforms.

The Java API is a large collection of ready-made software components that provide many useful capabilities, such as graphical user interface (GUI) widgets. The Java API is grouped into libraries **(packages)** of related components. The following figure depicts a Java program, such as an application or applet, that's running on the Java platform. As the figure shows, the Java API and Virtual Machine insulates the Java program from hardware dependencies.

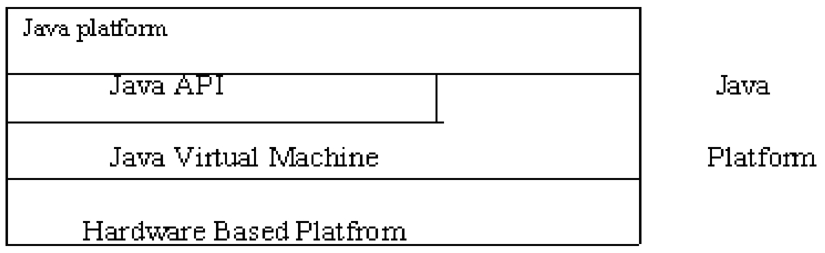

Fig. 1.4 JAVA Platform

As a platform-independent environment, Java can be a bit slower than native code. However, smart compliers, wheel - tuned interpreters, and just-in-time byte compilers can bring Java's performance close to that of native code without threatening portability.

Apache Tomcat Server

Apache Tomcat (formerly under the Apache Jakarta Project; Tomcat is now a top - level project) is a web container developed at the Apache Software Foundation. Tomcat implements the servlet and the Java Server Pages (JSP) specifications from Sun Microsystems, providing an environment for Java code to run in cooperation with a web server. It adds tools for configuration and management but can also be configured by editing configuration files that are normally XML-formatted. Because Tomcat includes its own HTTP server internally, it is also considered a standalone web server.

Environment

Tomcat is a web server that supports servlets and JSPs. Tomcat comes with the Jasper compiler that compiles JSPs into servlets. The Tomcat servlet engine is often used in combination with an Apache web server or other web servers. Tomcat can also function as an independent web server. Earlier in its development, the perception existed that standalone Tomcat was only suitable for development environments and other environments with minimal requirements for speed and transaction handling. However, that perception no longer exists; Tomcat is increasingly used as a standalone web server in high-traffic, high-availability environments. Since its developers wrote Tomcat in Java, it runs on any operating system that has a JVM.

Product features

Tomcat 3.x (initial release)

- Implements the Servlet 2.2 and JSP 1.1 specifications
- Servlet reloading
- Basic HTTP functionality Tomcat 4.x
- Implements the Servlet 2.3 and JSP 1.2 specifications
- Servlet container redesigned as Catalina
- JSP engine redesigned as Jasper

- Coyote connector
- Java Management Extensions (JMX), JSP & Struts based administration

Tomcat 5.x

- Implements the Servlet 2.4 and JSP 2.0 specifications
- Reduced garbage collection, improved performance and scalability
- Native Windows and Unix wrappers for platform integration
- Faster JSP paring

Purpose

The performance of data processing systems based on system configuration, such as CPU, memory, network and storage. Advancements in storage have lagged due to limitations presented by latency and throughput. Modern techniques, such as in-memory databases, which rely on main memory for a data store medium, are faster than disk-optimized database systems, but are still limited by today's memory capability. In addition, in-memory database still lack a non-volatile storage medium to provide long-term persistent storage. To handle QF big data, SSD (solid state disks)-backed storage could be more efficient than HDD (hard disk drive)-backed storage.

Product Perspective

Cloud computing is a new form of internet-based computing that provides shared computer processing resources and data to computers and other devices on demand. It is the delivery of hosted services over the internet. Cloud computing services can be public, private or hybrid. The growing industry of cloud has provided a service paradigm of storage / computation outsourcing helps to reduce users' burden of IT infrastructure maintenance, and reduce the cost for both the enterprises and individual users. The three main benefits of cloud computing are self-service provisioning, elasticity, pay per use. The three broad categories of cloud computing are Infrastructure as a Service, Platform as a Service and Software as a Service.

CHAPTER 2

LITERATURE SURVEY

Dynamic Data Slicing in Multi Cloud Storage using Cryptographic Technique

In the modern era of computing, secure data sharing has become one of the challenges when considering the adoption of multi-cloud storage services. It has become one of the essential services in cloud computing. Many advantages of multi-cloud storage attracts the individuals and organization to move their data from remote to cloud servers. Recently many Multi-cloud storage services have been proposed but most of them focus on the single specific organization and file formats. In addition to this most providers use attribute based encryption which encrypts only particular database fields which reduces the trust of the many individuals and organizations. The biggest challenge that the present business world faces in multi-cloud storage is that there is no single standard architecture and procedure that can meet the requirements of the individuals and organizations. In order to address this challenge, this paper presents an effective architectural framework with a standard algorithm which would enable to enhance the secure data sharing through dynamic index based cryptographic data slicing. The proposed model is suitable for decision making process for the individuals and organizations in the adoption of multi-cloud storage service based on trust

Design and Implementation of Secure data Cloud Storage using Encryption

In this paper we describe the design and development of a cloud computing based secure multi cloud data storage using encryption. This application uses multiple cloud storages, to cooperatively store and maintain the client's data. We use two mechanisms - Multi Agent system (MAS) and Data Encoding technique. These are combined together to give a new mechanism to provide data integrity and security for client's data in cloud storages. In this the admin role is to store the data which is uploaded by the clients. When the client wants to upload the file, he needs to login after he register his details. When the client is registered his details are stored in server. Now when the client uploads the data, the data is divided into sub parts by the third - party agent and then each part is stored redundantly into different multiple cloud storages. The data which is uploaded by the client can be only viewed by him. If any person other than the actual user tries to modify the file which is uploaded by the client, a third - party agent (TPA) module sends the alerts to the client informing him that the file is being tried for modification.

Enhanced Security for Multi-Cloud Storage using Cryptographic Data Splitting with Dynamic Approach

The use of cloud computing has increased rapidly in many organizations. Security is considered to be the most critical aspects in a cloud computing environment due to the sensitive information stored in the cloud for users. The goal of cloud Security is mainly focused on the issues related to the data security and privacy aspects in cloud computing. This multi cloud model which is based on partitioning of application system into Distinct clouds instead of using single cloud service such as in Amazon cloud service. This will discuss and present the Cryptographic data splitting with dynamic approach for securing information. The metadata information is stored in private cloud. This approach prevents the unauthorized data retrieval by Hackers and intruders. The results and implementation for the new proposed model is analysed, in relation to addressing the security factors in cloud computing.

Insider Threats and Cryptographic Techniques in Secure Information Management

This publication presents some techniques for insider threats and cryptographic protocols in secure processes. Those processes are dedicated to the information management of strategic data splitting. Strategic data splitting is dedicated to enterprise management processes as well as methods of securely storing and managing this type of data. Because usually strategic data are not enough secure and resistant for unauthorized leakage, we propose a new protocol that allows to protect data in different management structures. The presented data splitting techniques will concern cryptographic information splitting algorithms, as well as data sharing algorithms making use of cognitive data analysis techniques. The insider threats techniques will concern data reconstruction methods and cognitive data analysis techniques. Systems for the semantic analysis and secure information management will be used to conceal strategic information about the condition of the enterprise. Using the new approach, which is based on cognitive systems allow to guarantee the secure features and make the management processes more efficient.

CHAPTER 3

SYSTEM DESIGN

The system has two major parts, one is file chunking where one huge file is sliced into many and can be retrieved by merging those chunks. The other is encryption and decryption of those chunks during their upload and download respectively. The first part is explained below in Fig. 3.1.

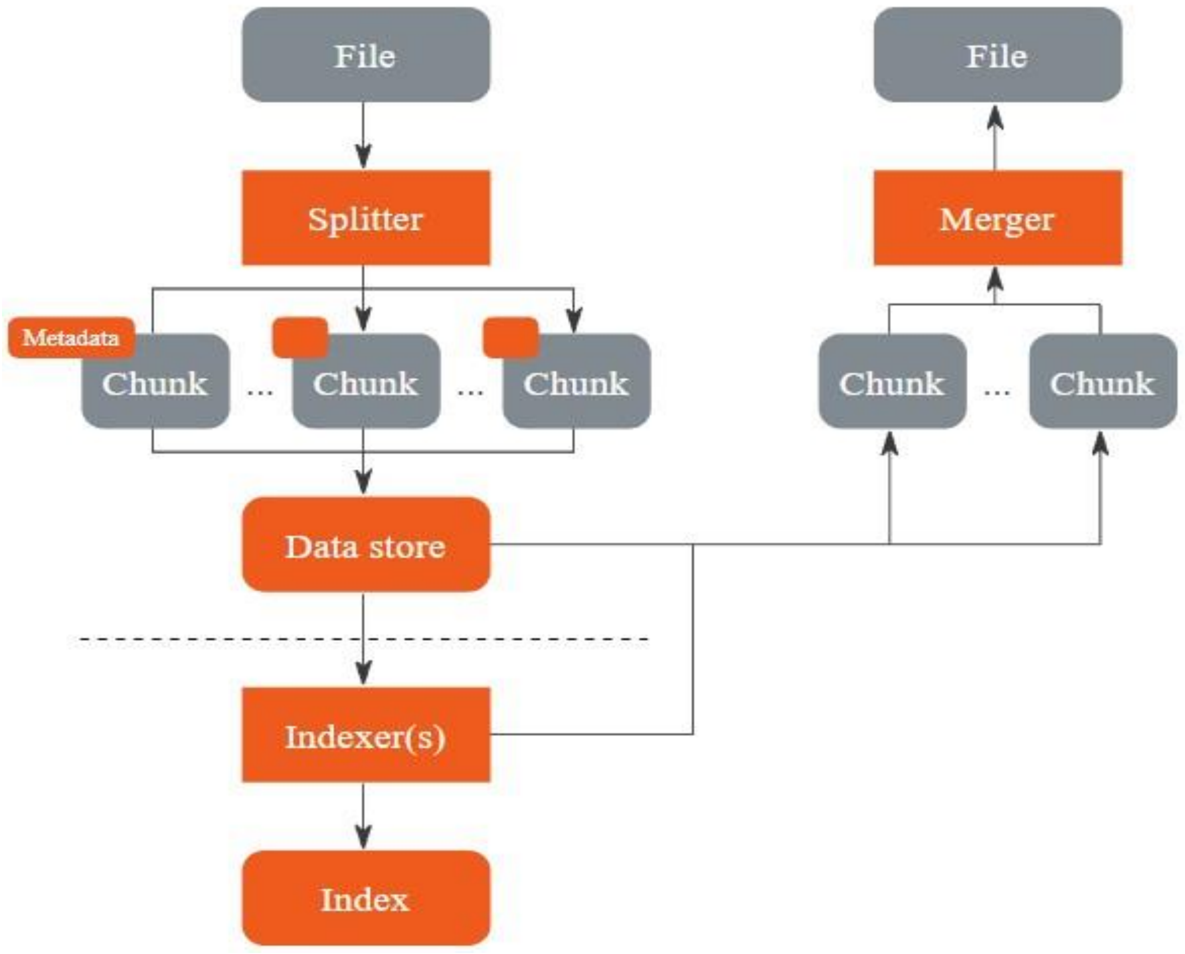

Fig. 3.1 File Splitting and Merging

An increment to the previous part, adding a layer of security i.e., Encryption and Decryption with the splitting and merging process respectively comes our second part as explained in Fig. 3.2

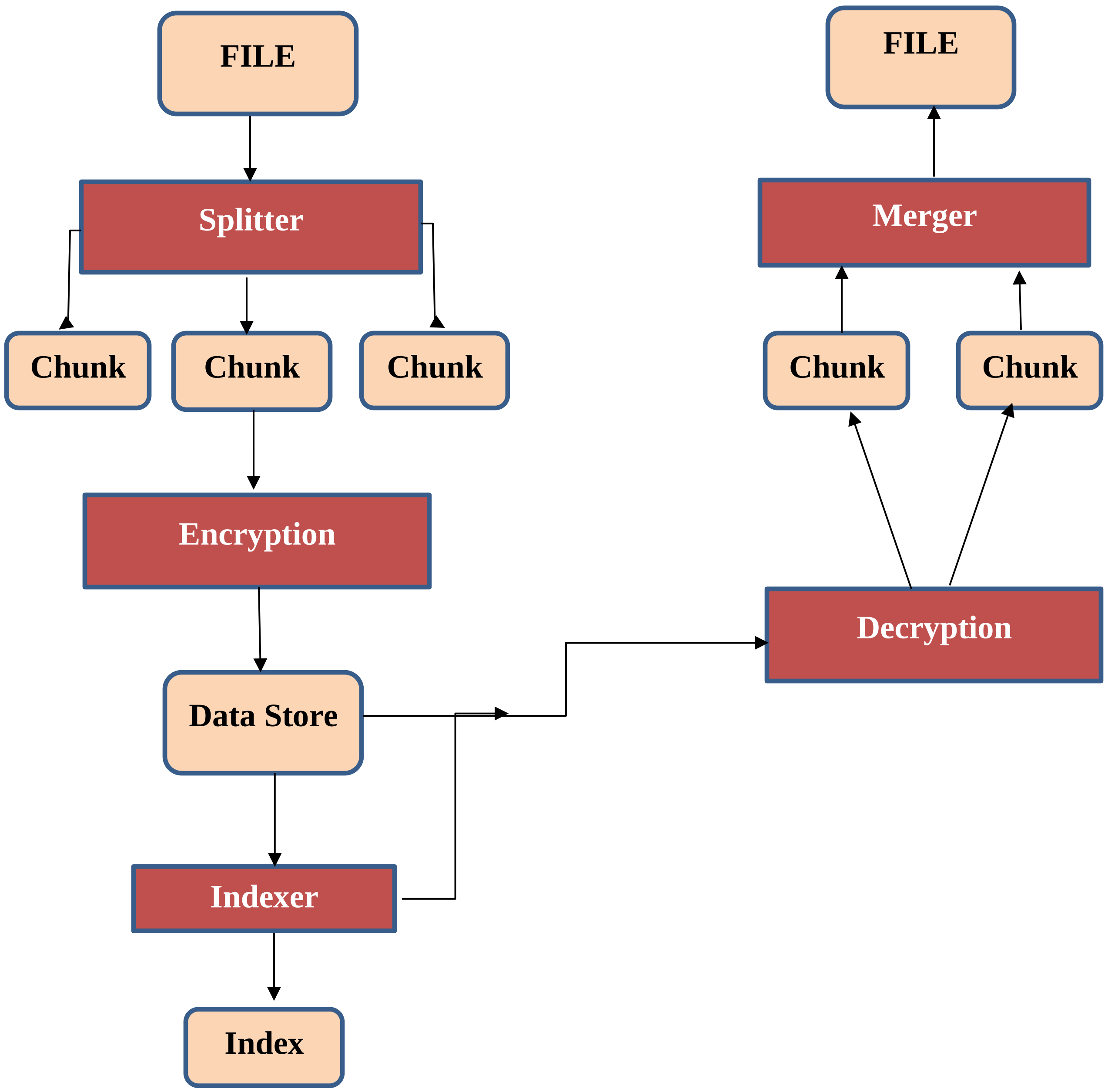

Fig. 3.2 Process with Encryption and Decryption

CHAPTER 4

ALGORITHM

The Encryption process involved in File security here is AES algorithm.

Advanced Encryption Standard

Through data sharing, higher productivity levels are reached. From the business point of view most cloud service provider offers only attribute based encryption. This type of encryption is based on few database fields such as account number, passwords etc. In attribute based encryption is used to protect the patient's information in health care organization. In order to reduce the cost of cloud hosting cloud storage providers prefer attribute based encryption.

AES Algorithm:

AES is based on a design principle known as a substitution - permutation network, and is efficient in both software and hardware. Unlike its predecessor DES, AES does not use a Feistel network. AES is a variant of Rijndael which has a fixed block size of 128 bits, and a key size of 128, 192, or 256 bits. By contrast, Rijndael *per se* is specified with block and key sizes that may be any multiple of 32 bits, with a minimum of 128 and a maximum of 256 bits. AES operates on a 4 × 4 column-major order array of bytes, termed the *state*. Most AES calculations are done in a particular finite field.

For instance, if there are 16 bytes, these bytes are represented as this two-dimensional array:

The key size used for an AES cipher specifies the number of transformation rounds that convert the input, called the plaintext, into the final output, called the ciphertext. The number of rounds are given as follows:

- 10 rounds for 128-bit keys.
- 12 rounds for 192-bit keys.
- 14 rounds for 256-bit keys.

Each round consists of several processing steps, including one that depends on the encryption key itself. A set of reverse rounds are applied to transform ciphertext back into the original plaintext using the same encryption key.

High-level description of the algorithm:

1. Key Expansion—round keys are derived from the cipher key using Rijndael's key schedule. AES requires a separate 128-bit round key block for each round plus one more.
2. Initial round key addition:
    1. Add Round Key—each byte of the state is combined with a block of the round key using bitwise xor.
3. 9, 11 or 13 rounds:
    1. SubBytes—a non-linear substitution step where each byte is replaced with another according to a lookup table.

2. ShiftRows—a transposition step where the last three rows of the state are shifted cyclically a certain number of steps.
3. MixColumns—a linear mixing operation which operates on the columns of the state, combining the four bytes in each column.
4. AddRoundKey

4. Final round (making 10, 12 or 14 rounds in total):
    1. SubBytes
    2. ShiftRows
    3. AddRoundKey

CHAPTER 5

SYSTEM REQUIREMENTS

5.1 INTRODUCTION

The requirement specification is a technical specification of the software and the hardware products required to build the application. It includes the functional, performance and security requirements. The purpose of mentioning software requirements specification is to provide a detailed overview of the application and its parameters, goals**.** It gives an idea about the target audience, the hardware and software suitable to run the application.

5.2 HARDWARE AND SOFTWARE REQUIREMENTS

5.2.1 Hardware Requirements

- Processor – Intel core i3
- Hard disk – 500GB min
- Ram – 4GB
- High speed Internet connection for upload and download
- Any client devices like mobile, Laptops, etc.

5.2.2 Software Requirements

- Windows 7
- Apache Tomcat 7 Server
- Modern Web browsers, IE7 or higher
- Java 7
- MySQL database 5

CHAPTER 6

IMPLEMENTATION

6.1 MODULES

1. **Sign up**
2. **Login**
3. **Upload**
4. **File Listing**
5. **Decrypt message**

6.2 MODULE EXPLANATION

6.2.1 Sign up

The sign up menu has three important fields. The user has to choose a username, password and conformation password. The values entered in these fields are stored in a database table. Which is retrieved when the same user logs in. MYSQL queries are used to get the values the user has entered and store it in the database.

6.2.2 Login

Users who have created an account can log in to proceed for encrypting or decrypting the message. This module has fields for username and password and it matched these credentials with the ones stored in the table. It uses secure channel to authenticate and validate the password.

6.2.3 Upload

Once the user has successfully logged in he/she can either choose the following option. In this module the user selects the file to be encrypted in the dialog box selection area provided. After selecting the file, the path is copied to the upload dialog box. The user must can choose any file format for Upload. After successful encryption of the file, the file is uploaded into the Cloud – Server. Here the splitting / Chunking process takes place and multiple folders are created and the file chunks are stored using RANDOM function. Similarly, the chunks are also renamed randomly. A Map file is generated, encrypted here and Downloaded to the Local storage.

6.2.4 File Listing

User files are listed in this module and the session logged-in user can access all of his uploaded files and can be processed, modified, deleted or even downloaded.

6.2.5 Download : Map File – Key Upload and Link Generation:

When the Map file is Uploaded the original file is generated. This process happens by the decipherment of the Map Key that was encrypted earlier. The chunks are combined and decrypted. Then the Download Link is generated here.

CHAPTER 7

CONCLUSION AND FUTURE WORK

7.1 CONCLUSION

The proposed application was built and for the purpose of securing the files that are stored in the cloud. The conclusion is that though we can strengthen the process of encryption and decryption, the safety of the file transferred is still a big issue to be solved. Hence, we opt to a few safer, but still not the most secure methods to protect our files for being accessed illegally.

7.2 FUTURE WORK

Asymmetric Key Encryption is one of the key factors of future work. Multiple cloud security methods and algorithms can be proposed. Digital Signature methods can also be included. Dynamic Encryption method can be designed for better access and security.

APPENDICES

APPENDIX – A (SAMPLE CODE)

Content:

-Encryption & Decryption

-Crypto.java

-Splitter

-Splitter.java

-Merger

-Merger.java

-ThreadProcess.java

Java codes:

Encryption & Decryption:

Crypto.java

```
package com.mini.crypt;
```

```
}

import java.io.IOException;
import java.io.OutputStream;
import java.security.InvalidAlgorithmParameterException;
import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;
import java.security.SecureRandom;
import java.security.spec.InvalidKeySpecException;
import java.security.spec.KeySpec;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.crypto.BadPaddingException;
import javax.crypto.NoSuchPaddingException;
import javax.crypto.Cipher;
import javax.crypto.IllegalBlockSizeException;
import javax.crypto.SecretKey;
import javax.crypto.SecretKeyFactory;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.PBEKeySpec;
import javax.crypto.spec.SecretKeySpec;

public class Crypto {

    String password;
    byte[] salt;
    byte[] iv;
    Cipher ci;

    public Crypto(String password){

        this.password = password;
    }
```

```
public byte[] getSalt(){

    return salt;
}

public byte[] getIv(){

    return iv;
}

public void encryptInit(){

    salt = new byte[8];
    SecureRandom srandom = new SecureRandom();
    srandom.nextBytes(salt);

    SecretKeySpec skey;
    SecretKeyFactory factory;

    iv = new byte[16];
    srandom.nextBytes(iv);
    IvParameterSpec ivspec = new IvParameterSpec(iv);

    try {

        factory = SecretKeyFactory.getInstance("PBKDF2WithHmacSHA256");
        KeySpec spec = new PBEKeySpec(password.toCharArray(), salt, 10000, 128);
        SecretKey tmp = factory.generateSecret(spec);
        skey = new SecretKeySpec(tmp.getEncoded(), "AES");
```

```
            ci = Cipher.getInstance("AES/CBC/PKCS5Padding");
            ci.init(Cipher.ENCRYPT_MODE, skey, ivspec);

        }
        catch (NoSuchAlgorithmException | InvalidKeySpecException | NoSuchPaddingException
| InvalidKeyException | InvalidAlgorithmParameterException ex) {
        }
    }

    public void decryptInit(byte[] salt, byte[] iv){

        try {

            SecretKeyFactory factory =
SecretKeyFactory.getInstance("PBKDF2WithHmacSHA256");
            KeySpec spec = new PBEKeySpec(password.toCharArray(), salt, 10000, 128);
            SecretKey tmp = factory.generateSecret(spec);
            SecretKeySpec skey = new SecretKeySpec(tmp.getEncoded(), "AES");
            IvParameterSpec ivspec = new IvParameterSpec(iv);

            ci = Cipher.getInstance("AES/CBC/PKCS5Padding");
            ci.init(Cipher.DECRYPT_MODE, skey, ivspec);

        }
        catch (NoSuchAlgorithmException | InvalidKeySpecException | NoSuchPaddingException
| InvalidKeyException | InvalidAlgorithmParameterException ex) {
        }
    }

    public void encryptFile(byte[] ibuf, OutputStream out){
```

```
    try {

        byte[] obuf = ci.doFinal(ibuf);
        if ( obuf != null ) out.write(obuf);

        out.flush();
        out.close();

    } catch (IOException | IllegalBlockSizeException | BadPaddingException ex) {
        Logger.getLogger(Crypto.class.getName()).log(Level.SEVERE, null, ex);
    }
}

public byte[] decryptFile(byte[] ibuf){

    try{

        byte[] obuf = ci.doFinal(ibuf);
        System.out.println("Decrypted");
        if(obuf != null) return obuf;
    }
    catch(BadPaddingException | IllegalBlockSizeException e){

        System.out.println(e);
        System.out.println("Error at decryption");
        return null;
    }

    System.out.println("Null is returned");
    return null;
}
```

```
}
```

Splitter:

Splitter.java

```
package com.mini.process;

import com.mini.crypt.Crypto;
import com.mini.dao.FilesDBdao;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.Random;

//Splits the uploaded file, encrypts and store them in the server
//Generates map file and add it to Database

public class Splitter {

    File file;
    String fileName;
    int user_id;
    String ip;
    String encKey;
    public static int fixed = 10485760;
    static byte nl[] = System.getProperty("line.separator").getBytes();
    Splitter(File file, String fileName, int user_id, String ip, String encKey) {
        this.file = file;
        this.fileName = fileName;
        this.user_id = user_id;
        this.ip=ip;
        this.encKey = encKey;

    }

    public void splitFile() throws IOException {

        byte buffer[] = new byte[fixed];

        FileOutputStream mos;

        String chunkPath = ip + File.separator + "chunks";
        String mapPath = ip + File.separator + "Maps";
        File pathv = new File(chunkPath);

        if(!pathv.exists()) pathv.mkdirs();
        pathv = new File(mapPath);

        if(!pathv.exists()) pathv.mkdirs();

        try (FileInputStream fis = new FileInputStream(file)) {

            int i = 1;
            File mapFile = new File(mapPath + File.separator + fileName.substring(0,
fileName.lastIndexOf('.')) + ".map");
            mapFile.createNewFile();
            mos = new FileOutputStream(mapFile, true);
            System.out.println("Password is " + encKey);

            Crypto crypto = new Crypto(encKey);
            crypto.encryptInit();

            mapInit(mos, crypto);

            while (fis.read(buffer) != -1) {

                String name = uniqueName(chunkPath);
                String chunk = chunkPath + File.separator + name;
                File f = new File(chunk);
                f.createNewFile();
                try (FileOutputStream fos = new FileOutputStream(f)) {
                    crypto.encryptFile(buffer, fos);
                    mos.write((i + "@" + name).getBytes());
                    mos.write(nl);
                    i++;
                    buffer = new byte[fixed];
                }
            }
            FilesDBdao.updateDB(mapFile.getName(), mapFile.getName(), user_id, file.length());
        }
        mos.close();
    }

    static String getAlphaNumericString(int n){
        //Get random alpha numeric string
        int lowerLimit = 97;
        Random random = new Random();
        StringBuilder r = new StringBuilder(n);

        for (int i = 0; i < n; i++) {
            int nextRandomChar = lowerLimit + (int)(random.nextFloat() * 26);
            r.append((char)nextRandomChar);
        }
        return r.toString();
    }

    static String uniqueName(String path){
        //Get unique name
        String chunkName = getAlphaNumericString(20);
        String newpath = path + File.separator + chunkName;

        while((new File(newpath)).exists()){
            chunkName = getAlphaNumericString(20);
            newpath = path + File.separator + chunkName;
        }
        return chunkName;
    }

    void mapInit(FileOutputStream mos, Crypto crypto) throws IOException{

        mos.write(fileName.getBytes());
        mos.write(nl);
        mos.write(crypto.getSalt());
        mos.write(nl);
        mos.write(crypto.getIv());
        mos.write(nl);
    }
```

```
}
```

Merger:

Merger.java

```
package com.mini.download;

import com.mini.crypt.Crypto;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.util.ArrayList;

/**
 *
 * @author Prakash
 */
public class Merger {

    static ArrayList<String>mapList;

    public static String merge(String mapPath, String appPath, String deckey) throws
FileNotFoundException, IOException{

        File mapFile = new File(mapPath);
        String downloadName;
String fname = getNameFromMap(mapFile);
byte[] salt = getSalt(mapFile);
byte[] iv = getIv(mapFile);

Crypto crypto = new Crypto(deckey);

if(salt == null) System.out.println("Salt is null");

if(iv == null) System.out.println("iv is null");

crypto.decryptInit(salt, iv);

String downloadPath = appPath + File.separator + "Merged";

File mergLoc = new File(downloadPath);
if(!mergLoc.exists()){
   mergLoc.mkdirs();
}

downloadName = fname;
getMapList(mapFile, appPath);
downloadPath += File.separator + downloadName;
File file = new File(downloadPath);

new PrintWriter(file).close();
file.deleteOnExit();

int size = mapList.size();
ThreadProcess tp[] = new ThreadProcess[size];
int i = 0;
```

```
    for(String s : mapList){
        tp[i] = new ThreadProcess(file, s, deckey, crypto);
        i++;
    }

    for (int k = 0; k < i; k += 8) {
        int t = k + 8;
        for (int j = k; j < t && j < i; j++) {
            tp[j].start();
        }

        for (int j = k; j < i && j < t; j++) {
            try {
                tp[j].join();
            }
            catch (InterruptedException e) {
            }
        }
        for (int j = k; j < i && j < t; j++) {
            try {
                tp[j].writeToFile();
            }
            catch (IOException e) {
            }
        }
    }

    return downloadName;
}
```

```
    public static byte[] getSalt(File map){

        if (map.exists()) {
            try (FileInputStream fis = new FileInputStream(map);
                    BufferedReader br = new BufferedReader(new InputStreamReader(fis))) {

                br.readLine();
                byte[] salt = new byte[8];
                fis.read(salt);
                return salt;
            }
            catch (IOException e) {
                System.out.println(e);
            }
        }
        return null;
    }
}
```

APPENDIX - B (SCREEN SHOTS)

Login Page

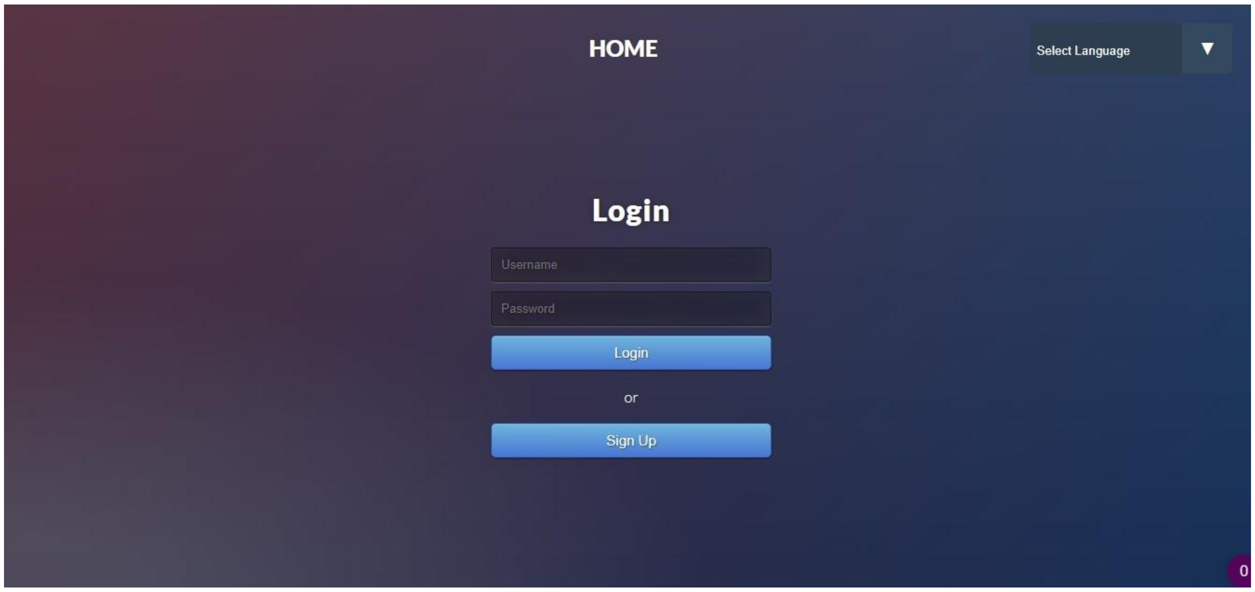

Fig. A i) Login Screen

Sign Up

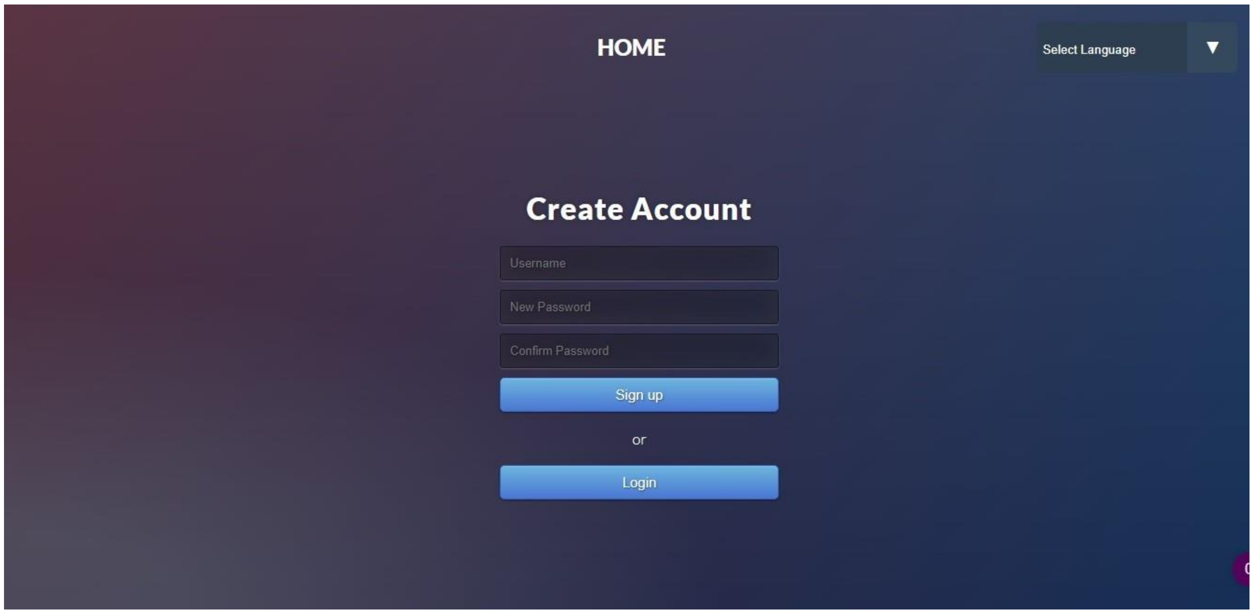

Fig. A ii) Signup Screen

Shared File Module

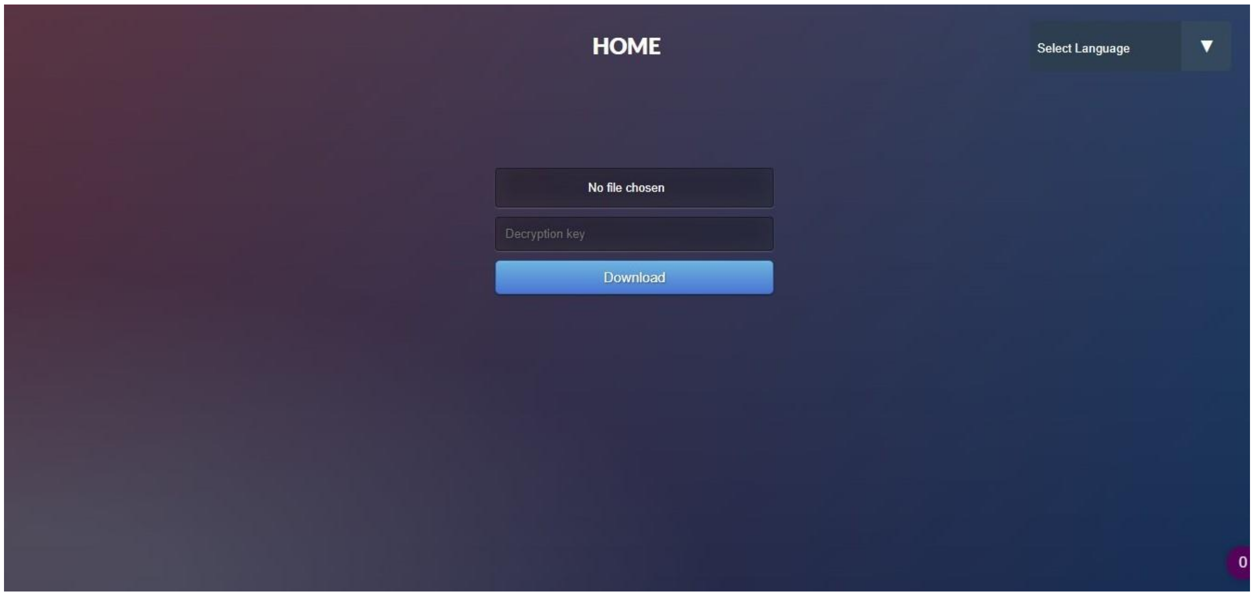

Fig. A iii) Map file Upload and Decryption Page

Upload file Module

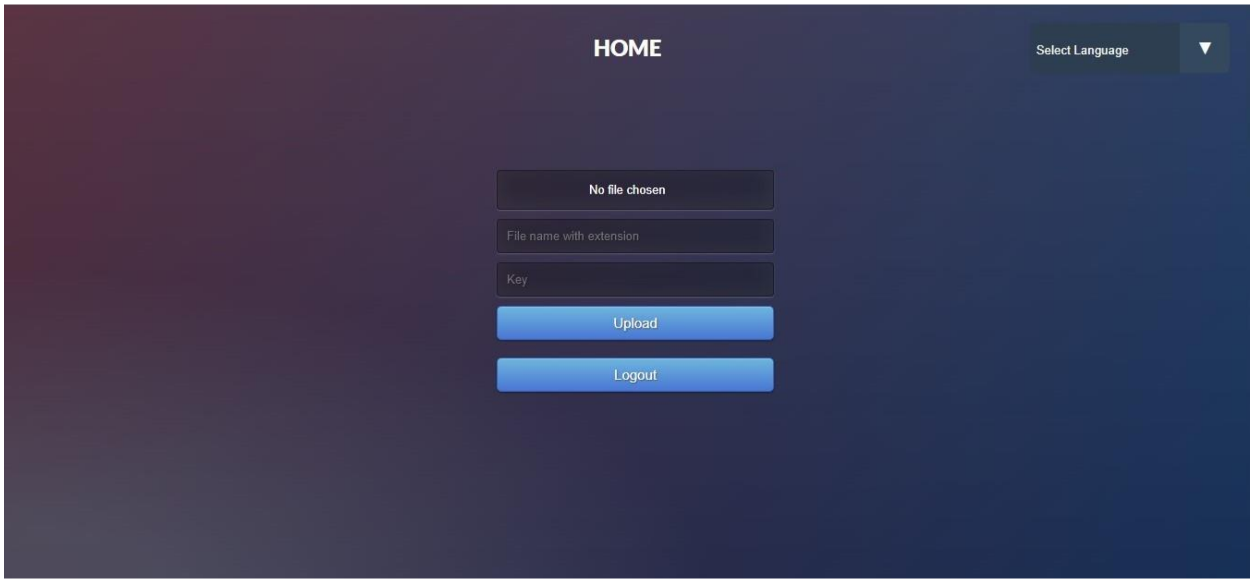

Fig. A iv) File Upload Page

REFERENCES

1. K. Subramanian, F. Leo John, "Dynamic Data Slicing in Multi Cloud Storage Using Cryptographic Technique", Computing and Communication Technologies (WCCCT) 2017 World Congress on, pp. 159-161, 2017.
2. V.R. Balasaraswathi, S. Manikandan, "Enhanced security for multi-cloud storage using cryptographic data splitting with dynamic approach" in Advanced Communication Control and Computing Technologies (ICACCCT) 2014 International Conference on, IEEE, pp. 1190-1194, 2014.’
3. B. Fabian, T. Ermakova, P. Junghanns, "Collaborative and secure sharing of healthcare data in multi-clouds", Information Systems, vol. 48, pp. 132-150, 2015.
4. Ali Mazhar, Dhamotharan Revathi, Khan Eraj, U. Khan Samee, V. Vasilakos Athanasios, "SeDaSC: Secure Data Sharing in Clouds" in Systems Journal, IEEE, pp. 1-10, 2014.
5. WANG Liang-Liang, CHEN Ke-Fei, MAO Xian-ping, WANG Yong-Tao, Efficient and Provably-Secure Certificateless Proxy Re-encryption Scheme for Secure Cloud Data Sharing Journal of Shanghai Jiaotong University (2014), vol. 19, no. 4, pp. 398-405
6. Tatiana Ermakova, "Benjamin Fabian Secret Sharing for Health Data in Multi-provider Clouds Business Informatics (CBI)", 2013 IEEE 15th Conference, pp. 93-100, 2013.

7. S. H. Seo, M. Nabeel, X. Ding, E. Bertino, "An efficient certificateless encryption for secure data sharing in public clouds", Knowledge and Data Engineering IEEE Transactions on, vol. 26, no. 9, pp. 2107-2119, September 2013.
8. Peng Xul, Xiaqi Liu, Zhenguo Sheng, Xuan Shan', Kai Shuang, "SSDS-MC: Slice-based Secure Data Storage in MultiCloud Environment 11 th EAI International Conference on Heterogeneous Networking for Quality", Reliability Security and Robustness (QSHINE 2015), pp. 304-309.
9. Yuuki Kajiura, Shohei Ueno, Atsushi Kanai, Shigeaki Tanimoto, Hiroyuki Sato, "An Approach to Selecting Cloud Services for Data Storage in Heterogeneous-Multicloud Environment with High Availability and Confidentiality", Autonomous Decentralized Systems (ISADS 2015) IEEE Twelfth International Symposium, pp. 205-210
10. A. N. Khan, M. M. Kiah, S. A. Madani, M. Ali, S. Shamshirband, "Incremental proxy re-encryption scheme for mobile cloud computing environment", The Journal of Supercomputing, vol. 68, no. 2, pp. 624-651, May 2014.

PUBLICATION

TECHNICAL PROJECT OUTCOME

After successful completion of project work student will be able to:

TPO1: Analyze, Design and Implement projects with a comprehensive, systematic and ethical approach.

TPO2: Apply modern tools to execute and integrate modules in the project.

TPO3: Apply techniques for societal, health care, and real time sustainable research projects.

TPO4: Develop communication skills by the technical presentation activities.

TPO5: Contribute as a team and lead the team in managing technical projects.

Mapping of Technical Project outcomes with the Project

	TPO1	**TPO2**	**TPO3**	**TPO4**	**TPO5**
Project Title	3	3	2	3	3

(Indicate as 1 – Less than 30%; 2 – 30.1 - 60% and 3 –above 60.1%)