

## **1. SYNOPSIS**

As the result of growth in technology utilization, the usage of data has increased. Transfer of data from one system to other system is made possible with new technologies. This results in the problems of security and dependability. In order to provide efficient transfer of data without allowing intruders to guess data, security plays the important role. Dependability is to recover original data in case of any errors. In order to achieve these two properties several algorithms came in existence. Usage of these algorithms results in fast and secure transfer of data

By using RSA algorithm the input file is encrypted and the encrypted file is compressed by using LZW compression algorithm. The resultant file is transferred to client system along with keys which include private key of RSA algorithm. After receiving files, client will execute LZW decompression algorithm to retrieve encrypted file, Later by executing RSA decryption algorithm the original file is retrieved. By using these two algorithms we can achieve security and as well as faster transmission.

## **2. SYSTEM CONFIGURATION**

### **HARDWARE REQUIREMENTS**

The hardware requirements for the current system are

- Minimum of two systems one for server and other for client
- Minimum of 2GB RAM.
- Minimum 80GB hard-disk
- Minimum dual core processor.

### **SOFTWARE REQUIREMENTS**

The software requirements for the current system are

- Windows Operating System( can be either windows 7/8)
- Java Development Environment

### 3. OVERVIEW OF PROJECT PLATFORM

The platforms used are:

#### **Java:**

Java is a set of several computer software products and specifications from Sun Microsystems (which has since merged with Oracle Corporation), that together provide a system for developing application software and deploying it in a cross platform computing environment. Java is used in a wide variety of computing platforms from embedded devices and mobile phones on the low end, to enterprise servers and supercomputers on the high end. While less common, Java applets are sometimes used to provide improved and secure functions while browsing the World Wide Web on desktop computers. Writing in the Java programming language is the primary way to produce code that will be deployed as Java Byte code. There are, however, byte code compilers available for other languages such as Ada, JavaScript, Python, and Ruby. Several new languages have been designed to run natively on the Java Virtual Machine (JVM), such as Scala, Clojure and Groovy. Java syntax borrows heavily from C and C++, but object-oriented features are modeled after Smalltalk and Objective-C. Java eliminates certain low-level constructs such as pointers and has a very simple memory model where every object is allocated on the heap and all variables of object types are references. Memory management is handled through integrated automatic garbage collection performed by the JVM

**Java Card:** A technology that allows small Java-based applications (applets) to be run securely on smart cards and similar small-memory devices.

**Java ME (Micro Edition):** Specifies several different sets of libraries (known as profiles) for devices with limited storage, display, and power capacities. Often used to develop applications for mobile devices, PDAs, TV set-top boxes, and printers.

**Java SE (Standard Edition):** For general-purpose use on desktop PCs, servers and similar devices.

Java EE (Enterprise Edition): Java SE plus various APIs useful for multi-tier client–server enterprise applications.

The Java platform consists of several programs, each of which provides a portion of its overall capabilities. For example, the Java compiler, which converts Java source code into Java byte code (an intermediate language for the JVM), is provided as part of the Java Development Kit (JDK). The Java Runtime Environment (JRE), complementing the JVM with a just-in-time (JIT) compiler, converts intermediate byte code into native machine code on the fly. An extensive set of libraries are also part of the Java platform. The essential components in the platform are the Java language compiler, the libraries, and the runtime environment in which Java intermediate byte code "executes" according to the rules laid out in the virtual machine specification

## **Windows:**

Microsoft Windows is a series of graphical interface operating systems developed, marketed, and sold by Microsoft. The early versions of Windows are often thought of as graphical shells, mostly because they ran on top of MS-DOS and use it for file system services. However, even the earliest Windows versions already assumed many typical operating system functions; notably, having their own executable file format and providing their own device drivers (timer, graphics, printer, mouse, keyboard and sound). Unlike MS-DOS, Windows allowed users to execute multiple graphical applications at the same time, through cooperative multitasking. Windows implemented an elaborate, segment-based, software virtual memory scheme, which allows it to run applications larger than available memory: code segments and resources are swapped in and thrown away when memory became scarce, data segments moved in memory when a given application had relinquished processor control. Windows NT included support for several different platforms before the x86-based personal

computer became dominant in the professional world. Windows NT 4.0 and its predecessors supported PowerPC, DEC Alpha and MIPS R4000. Microsoft released Windows XP Professional x64 Edition and Windows Server 2003 x64 Editions to support the x86-64 (or simply x64), the eighth generation of x86 architecture. An edition of Windows 8 known as Windows RT was specifically created for computers with ARM architecture. Consumer versions of Windows were originally designed for ease-of-use on a single-user PC without a network connection, and did not have security features built in from the outset. However, Windows NT and its successors are designed for security (including on a network) and multi-user PCs. Windows 7 was intended to be a more focused, incremental upgrade to the Windows line, with the goal of being compatible with applications and hardware with which Windows Vista was already compatible. Windows 7 has multi-touch support, a redesigned Windows shell with an updated taskbar, a home networking system called Home Group, and performance improvements.

Windows 8, the successor to Windows 7, was released generally on October 28, 2012. A number of significant changes were made on Windows 8, including the introduction of a user interface based around Microsoft's Metro design language with optimizations for touch-based devices such as tablets and all-in-one PCs. These changes include the Start screen, which uses large tiles that are more convenient for touch interactions and allow for the display of continually updated information, and a new class of apps which are designed primarily for use on touch based devices. Other changes include increased integration with cloud services and other online platforms (such as social networks and Microsoft's own SkyDrive and Xbox Live services), the Windows Store service for software distribution, and a new variant known as Windows RT for use on devices that utilize the ARM

## **4. PROJECT DESCRIPTION**

Growth in technology results increase in usage of information electronically. Security plays a important role in transmitting of this type of information. The electronic information that is transmitted may contain some confidential message such that it should be accessed by trusted users alone and it should be prevented from others. There are several algorithms that are designed for providing security one such type of algorithm is RSA. Transmission of data is faster if data size is small. In order to reduce the size of the data, compression mechanisms plays a vital role, one such mechanism is LZW compression and decompression algorithm. By using these two algorithms (i-e RSA and LZW) fast and secure data transmission is achieved. .

### **Server Deployment**

One system is made as server which contains input file. Within that system, code for RSA encryption and LZW compression is written and executed. Server contains socket programming which transfer files to client by knowing clients IP address. Two socket programs are written one for sending encrypted and compressed file and the other for sending keys that are used for decrypting the encrypted file.

### **Client Deployment**

The target system is made as client which acts as receiver. Within the client system code for RSA decryption and LZW decompression algorithms are written and executed. Similar to server two socket programs are written for receiving the files from the server. These files are given as input to LZW decompression and RSA decryption algorithms.

## **RSA Encryption**

RSA is an algorithm for public-key cryptography that is based on the presumed difficulty of factoring large integers, the factoring problem. RSA stands for Ron Rivest, Adi Shamir and Leonard Adleman, It is the encryption algorithm which uses two keys one is public key and other is private key. Public key is used to encrypt the original file, whereas private key is used for decryption purpose. A user of RSA creates and then publishes the product of two large prime numbers, along with an auxiliary value, as their public key. The prime factors must be kept secret. Anyone can use the public key to encrypt a message. Messages encrypted with the public key can only be decrypted in a reasonable amount of time using the private key. The keys for the RSA algorithm are generated the following way:

- Choose two distinct prime numbers  $p$  and  $q$
- Compute  $n = pq$ .
- Compute  $\phi(n) = \phi(p)\phi(q) = (p - 1)(q - 1)$
- Choose an integer  $e$  such that  $1 < e < \phi(n)$  and  $\gcd(e, \phi(n)) = 1$ .  $e$  is public key
- Determine  $d$  as  $d^{-1} \equiv e \pmod{\phi(n)}$  here  $d$  is private key

By using  $c = m^e \pmod{n}$  we can convert ASCII value to cipher value and is written to a file. This process is continued until the end of file is reached. The private key and  $n$  value are written to other file in order to send it to client for decryption purpose. The encrypted file is used as input for compression algorithm.

## **LZW Compression**

LZW compression code receives encrypted file as input. It creates a dictionary which contains ASCII values of all 256 characters. A dictionary is initialized to contain the single-character strings corresponding to all the possible input characters (and nothing else except the clear and stop codes if

they're being used). The algorithm works by scanning through the input string for successively longer substrings until it finds one that is not in the dictionary. When such a string is found, the index for the string without the last character (i.e., the longest substring that *is* in the dictionary) is retrieved from the dictionary and sent to output, and the new string (including the last character) is added to the dictionary with the next available code. The last input character is then used as the next starting point to scan for substrings. In this way, successively longer strings are registered in the dictionary and made available for subsequent encoding as single output values. The algorithm works best on data with repeated patterns, so the initial parts of a message will see little compression. As the message grows, however, the compression ratio tends asymptotically to the maximum. This encoded file is sent to client by using socket programming and keys are also sent by using other socket program

### **LZW Decompression**

The decoding algorithm works by reading a value from the encoded input and outputting the corresponding string from the initialized dictionary. At the same time it obtains the next value from the input, and adds to the dictionary the concatenation of the string just output and the first character of the string obtained by decoding the next input value, or the first character of the string just output if the next value cannot be decoded (If the next value is unknown to the decoder, then it has just been added, and so its first character must be the same as the first character of the string just output). The decoder then proceeds to the next input value (which was already read in as the "next value" in the previous pass) and repeats the process until there is no more input, at which point the final input value is decoded without any more additions to the dictionary. In this way the decoder builds up a dictionary which is identical to that used by the encoder, and uses it to decode subsequent input values. Thus the full dictionary does not need be sent with the encoded data; just the initial dictionary containing the



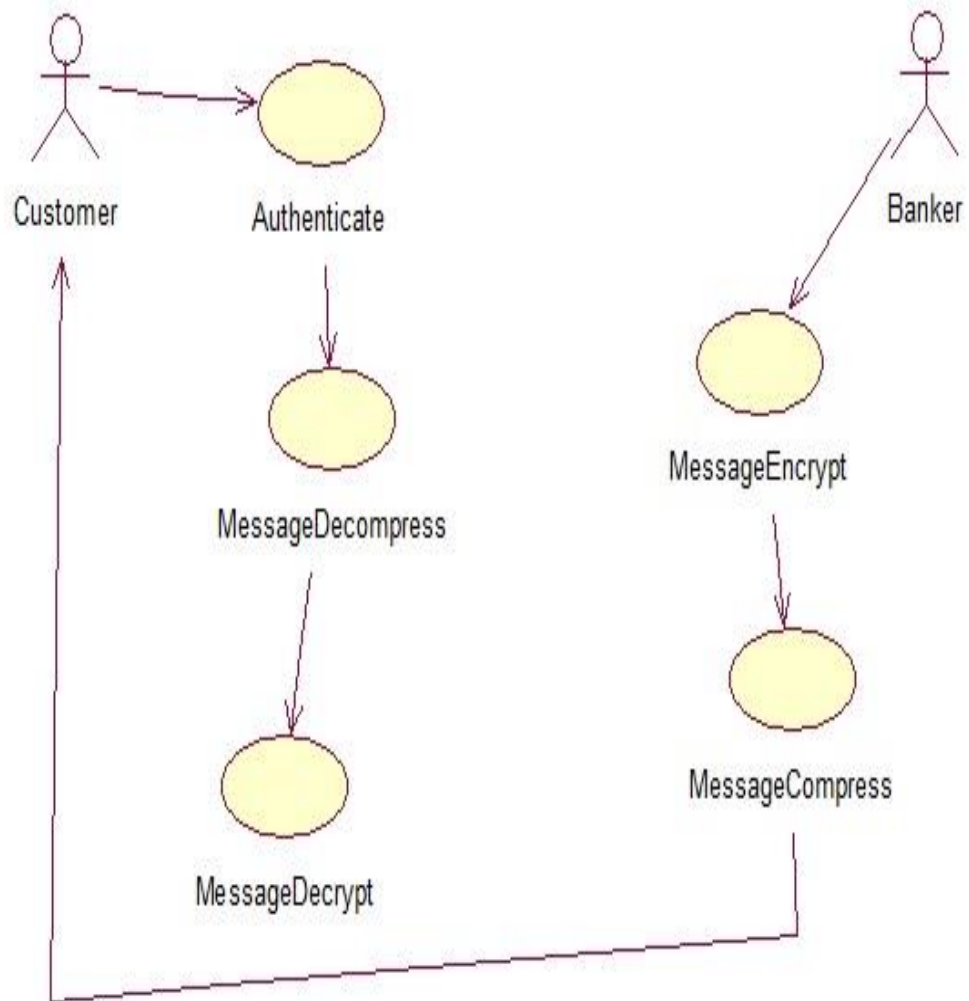
single-character strings is sufficient (and is typically defined beforehand within the encoder and decoder rather than being explicitly sent with the encoded data.)

## **RSA Decryption**

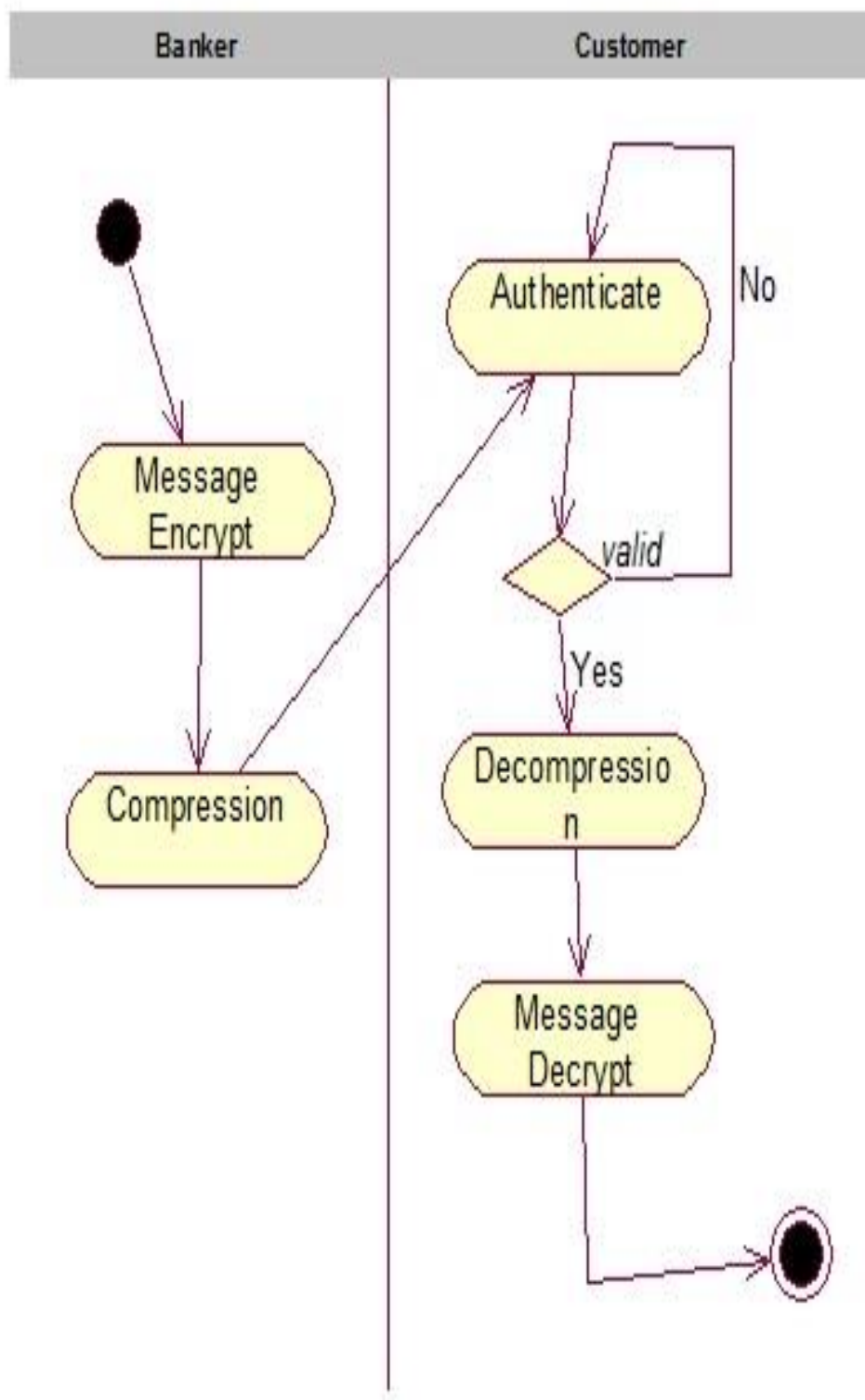
RSA Decryption code will take decompressed file as input which contains cipher values. It receives private key generated by RSA encrypted file in server through socket programming. By using this private key the cipher values are converted to plain values. These plain values are generated by using the formula  $m \equiv c^d \pmod{n}$  where  $c$  is the value obtained from the file,  $d$  is the private key and  $n$  is the product of two prime numbers. This process is repeated until the end of file is reached. These plain values are written to output file. Output file will contains the information exactly as in that of input file in server. In order to make the algorithm work more effectively a cryptographically strong random number generator, which has been properly seeded with adequate entropy, must be used to generate the primes  $p$  and  $q$ . This results in fast and secure transmission of data from server to client

## 4. Diagrams

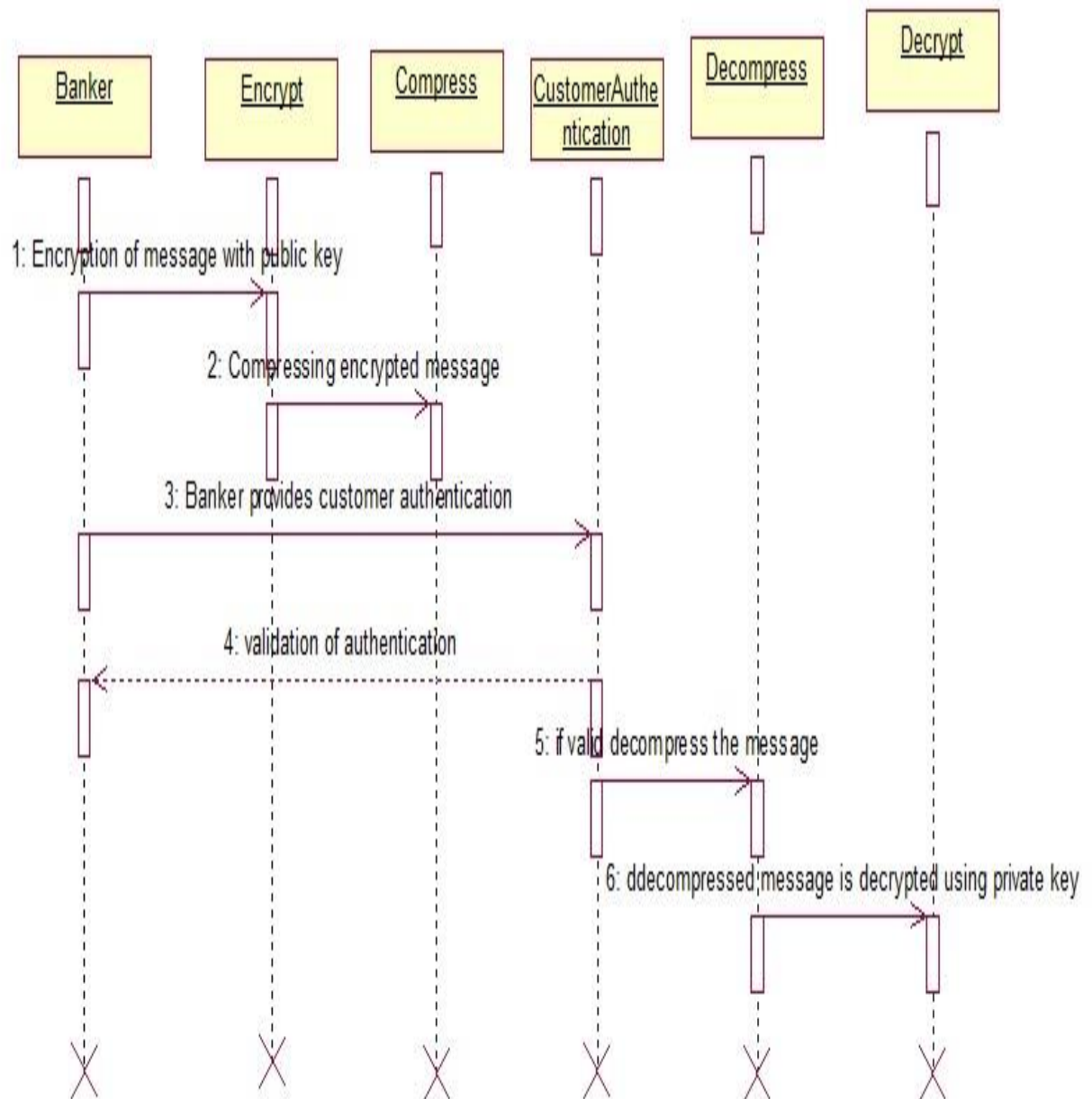
### Use Case Diagram for Entire System



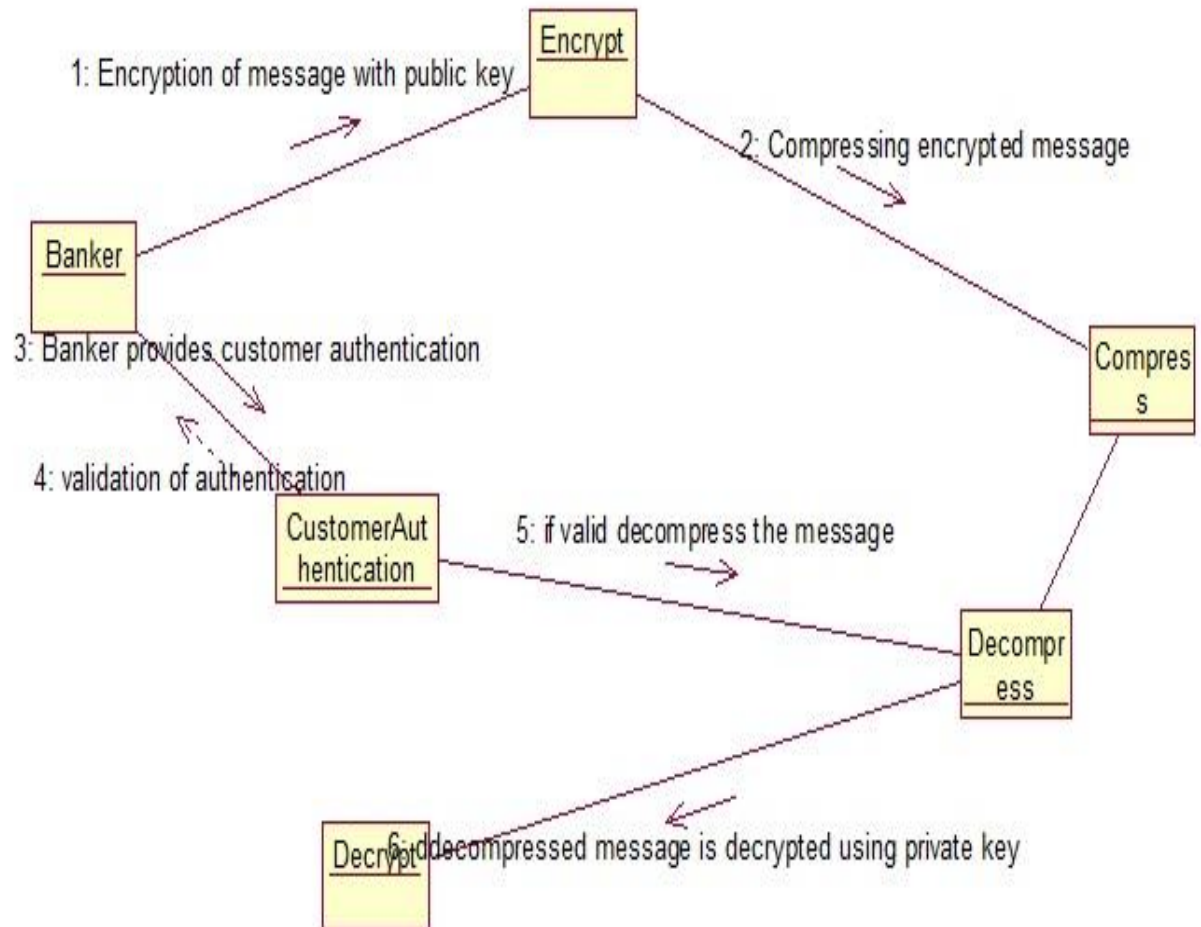
## Activity Diagram for Entire System



## Sequence Diagram for Entire System



## Collaboration Diagram for Entire system



## 5. Source Code

### RSA Encryption

#### **RSA.java**

```
import java.io.*;
import java.util.*;
import java.math.BigInteger;
class RSA
{
    public static void main(String s[]) throws IOException
    {

        DataInputStream ds=new DataInputStream(System.in);
        Random rand1=new Random();
        Random rand2=new Random();
        BigInteger p=BigInteger.probablePrime(16,rand1);
        BigInteger q=BigInteger.probablePrime(16,rand2);
        BigInteger r=p.multiply(q);
        BigInteger p1=p.subtract(new BigInteger("1"));
        BigInteger q1=q.subtract(new BigInteger("1"));
        BigInteger r1=p1.multiply(q1);

        System.out.println("Enter some sample public key value:");
        int pkey=Integer.parseInt(ds.readLine());
        while(true)
        {
            BigInteger numgcd=r1.gcd(new BigInteger(""+pkey));
            if(numgcd.equals(BigInteger.ONE))
                break;
            pkey++;
        }

        BigInteger pubkey=new BigInteger(""+pkey);
        BigInteger prvkey=pubkey.modInverse(r1);

        try
        {
            FileWriter fw=new FileWriter("keys.txt");
            FileWriter fw1=new FileWriter("rsaoutput.txt");
            String m=String.valueOf(prvkey);
            fw.write(m);
            fw.write("::");
            String n=String.valueOf(r);
            fw.write(n);
            fw.close();
            FileInputStream fin=new FileInputStream("rsainput.txt");
            int c;
            while((c=fin.read())!=-1)
```

```

        {
            BigInteger val=new BigInteger(""+c);
            BigInteger cival=val.modPow(pubkey,r);
            String s1=String.valueOf(cival);
            fw1.write(s1);
            fw1.write("xx");
        }
        fw1.close();
        System.out.println("\n\nENCRYPTION IS SUCCESSFULL\n\n");
    }
    catch(Exception e)
    {
    }
}
}
}

```

## **LZW Compression**

### **LZW.java**

```

import java.io.*;
import java.lang.Object;
import java.io.ObjectOutputStream;
import java.util.*;
import java.net.*;

class LZWCOMP
{
    public static int dictionary[][]=new int[10000][10000];
    public static int pattern[]=new int[10000];
    public static int dfree,psize;
    public static void ini()
    {
        for(int i=0;i<10000;i++)
        {
            for(int j=0;j<10000;j++)
                dictionary[i][j]=-1;
        }
        for(int i=0;i<10000;i++)
        {
            pattern[i]=-1;
        }
        for(int i=48;i<=58;i++)
        {
            dictionary[i][0]=i;
        }
    }
}

```

```

        dfree=59;
        psize=0;
    }

    public static boolean indict(int x)
    {
        pattern[psize]=x;
        psize++;
        for(int i=0;i<10000;i++)
        {
            if(pattern[0]==dictionary[i][0])
            {
                int n=1;
                for(int j=1;j<psize;j++)
                {
                    if(pattern[j]==dictionary[i][j])
                        n++;
                }
                if(n==psize)
                    return true;
            }
        }
        pattern[psize-1]=-1;
        psize--;
        return false;
    }

    public static void index(DataOutputStream out)throws IOException
    {
        for(int i=0;i<10000;i++)
        {
            if(pattern[0]==dictionary[i][0])
            {
                int n=1;
                for(int j=1;j<psize;j++)
                {
                    if(pattern[j]==dictionary[i][j])
                        n++;
                }
                if(n==psize)
                {
                    if(i>58)
                    {
                        int k=0;
                        out.write(244);
                        while(i>58)
                        {
                            i-=58;
                            k++;
                        }
                    }
                }
            }
        }
    }

```



```

        }
        out.write(k);
    }
    out.write(i);
    return;
}
}
}

public static void add(int x)
{
    for(int i=0;i<psize;i++)
        dictionary[dfree][i]=pattern[i];
    dictionary[dfree][psize]=x;
    dfree++;
}

public static void clear()
{
    for(int i=0;i<psize;i++)
        pattern[i]=-1;
    psize=0;
}

public static void compress(String infile,String outfile)
{
    try
    {
        DataInputStream in=new DataInputStream(new FileInputStream(infile));
        DataOutputStream out=new DataOutputStream(new FileOutputStream(outfile));
        int x=in.read();
        while(x!=-1)
        {
            if(indict(x));
            else
            {
                index(out);
                add(x);
                clear();
                psize=0;
                pattern[0]=x;
                psize++;
            }
            x=in.read();
        }
        index(out);
        in.close();
        out.close();
    }
}

```

```

    }

    catch(FileNotFoundException ex)
    {

    }

    catch(IOException ex)
    {

    }

}

public static void main(String args[])throws IOException
{
    String infile,outfile;
    infile="rsaoutput.txt";
    outfile="transfer.txt";
    ini();
    System.out.println("\n\ncompressing.....\n");
    compress(infile,outfile);
    System.out.println("\n\nCOMPRESSION IS DONE SUCCESSFULLY\n");

}
}

```

## **Data Transmission**

### **Server.java**

```

import java.net.*;
import java.io.*;
public class Server {

    public static void main (String [] args ) throws IOException {

        ServerSocket serverSocket = new ServerSocket(8080);
        Socket socket = serverSocket.accept();
        DataInputStream ds=new DataInputStream(System.in);
        System.out.println("Accepted connection : " + socket);
        String s="transfer.txt";
        File transferFile = new File (s);
        byte [] bytearray = new byte [(int)transferFile.length()];
        FileInputStream fin = new FileInputStream(transferFile);
        BufferedInputStream bin = new BufferedInputStream(fin);
    }
}

```

```

        bin.read(bytearray,0,bytearray.length);
        OutputStream os = socket.getOutputStream();
        System.out.println("Sending Files...");
        os.write(bytearray,0,bytearray.length);
        os.flush();
        socket.close();
        System.out.println("\n\nFile transfer complete\n\n");
    }
}

```

## Server2.java

```

import java.net.*;
import java.io.*;
public class Server2 {

    public static void main (String [] args ) throws IOException {

        ServerSocket serverSocket = new ServerSocket(8080);
        Socket socket = serverSocket.accept();
        DataInputStream ds=new DataInputStream(System.in);
        System.out.println("Accepted connection : " + socket);
        String s="keys.txt";
        File transferFile = new File (s);
        byte [] bytearray = new byte [(int)transferFile.length()];
        FileInputStream fin = new FileInputStream(transferFile);
        BufferedInputStream bin = new BufferedInputStream(fin);
        bin.read(bytearray,0,bytearray.length);
        OutputStream os = socket.getOutputStream();
        System.out.println("Sending Keys...");
        os.write(bytearray,0,bytearray.length);
        os.flush();
        socket.close();
        System.out.println("\n\nKey transfer complete\n\n");
    }
}

```

## Client.java

```

import java.net.*;
import java.io.*;
public class Client {

    public static void main (String [] args ) throws IOException {
        DataInputStream ds=new DataInputStream(System.in);
        int filesize=1022386;
        int bytesRead;
    }
}

```

```

int currentTot = 0;
Socket socket = new Socket("127.0.0.1",8080);
byte [] bytearray = new byte [filesize];
InputStream is = socket.getInputStream();
String s;
s="receive.txt";
FileOutputStream fos = new FileOutputStream(s);
BufferedOutputStream bos = new BufferedOutputStream(fos);
bytesRead = is.read(bytearray,0,bytearray.length);
currentTot = bytesRead;

do {
    bytesRead = is.read(bytearray, currentTot, (bytearray.length-currentTot));
    if(bytesRead >= 0)
        currentTot += bytesRead;
} while(bytesRead > -1);

bos.write(bytearray, 0 , currentTot);
System.out.println("\n\nFILE IS RECIEVED SUCCESSFULLY\n\n");
bos.flush();
bos.close();
socket.close();

}

}

```

### **Client2.java**

```

import java.net.*;
import java.io.*;
public class Client2 {

    public static void main (String [] args ) throws IOException {
        DataInputStream ds=new DataInputStream(System.in);
        int filesize=1022386;
        int bytesRead;
        int currentTot = 0;
        Socket socket = new Socket("127.0.0.1",8080);
        byte [] bytearray = new byte [filesize];
        InputStream is = socket.getInputStream();
        String s;
        s="keys.txt";
        FileOutputStream fos = new FileOutputStream(s);
        BufferedOutputStream bos = new BufferedOutputStream(fos);
        bytesRead = is.read(bytearray,0,bytearray.length);
        currentTot = bytesRead;

        do {

```

```

        bytesRead = is.read(bytearray, currentTot, (bytearray.length-currentTot));
        if(bytesRead >= 0)
            currentTot += bytesRead;
    } while(bytesRead > -1);

    bos.write(bytearray, 0 , currentTot);
    System.out.println("\n\nKEY IS RECIEVED SUCCESSFULLY\n\n");
    bos.flush();
    bos.close();
    socket.close();

}

}

```

## **LZW Decompression**

### **LZWDECOMP.java**

```

import java.io.*;
import java.lang.Object;
import java.io.ObjectOutputStream;
import java.util.*;
import java.net.*;

class LZWDECOMP
{
    public static int dictionary[][]=new int[8000][8000];
    public static int pattern[]=new int[8000];
    private static int element[]=new int[8000];
    public static int dfree,psize;
    public static int esize=0;
    public static void ini()
    {
        for(int i=0;i<8000;i++)
        {
            for(int j=0;j<8000;j++)
                dictionary[i][j]=-1;
        }
        for(int i=0;i<8000;i++)
        {
            pattern[i]=-1;
            element[i]=-1;
        }
        for(int i=48;i<=58;i++)
        {

```

```

        dictionary[i][0]=i;
    }
    dfree=59;
    psize=esize=0;
}
public static void clear()
{
    for(int i=0;i<psize;i++)
        pattern[i]=-1;
    psize=0;
}

public static void decompress(String infile,String outfile)
{
    try
    {
        DataInputStream in=new DataInputStream(new FileInputStream(infile));
        DataOutputStream out=new DataOutputStream(new FileOutputStream(outfile));
        int x=in.read();
        if(x!=-1)
        {
            element[0]=dictionary[x][0];
            esize++;
            out.write(dictionary[x][0]);
        }
        pattern[0]=element[0];
        psize++;
        while((x=in.read())!=-1)
        {
            if(x==244)
            {
                int k=in.read();
                int l=in.read();
                x=k*58+1;
            }
            if(x>=dfree)
            {
                for(int i=0;i<esize;i++)
                    element[i]=-1;
                esize=0;
                for(int i=0;i<psize;i++)
                {
                    element[i]=pattern[i];
                    esize++;
                }
            }
        }
    }
}

```

```

        element[esize]=pattern[0];
        esize++;
    }
    else
    {
        for(int i=0;i<esize;i++)
            element[i]=-1;
        esize=0;
        for(int i=0;i<8000;i++)
        {
            if(dictionary[x][i]!=-1)
            {
                element[i]=dictionary[x][i];
                esize++;
            }
            else
                break;
        }
    }
    for(int i=0;i<esize;i++)
        out.write(element[i]);
    for(int i=0;i<psize;i++)
        dictionary[dfree][i]=pattern[i];
    dictionary[dfree][psize]=element[0];
    dfree++;
    clear();
    for(int i=0;i<esize;i++)
    {
        pattern[i]=element[i];
        psize++;
    }
}
in.close();
out.close();
}

catch(FileNotFoundException ex)
{

}

catch(IOException ex)
{

}

```

```

    }

    public static void main(String args[])throws IOException
    {
        String infile,outfile;
        infile="receive.txt";
        outfile="decompress.txt";
        ini();
        decompress(infile,outfile);
        System.out.println("\n\nDECOMPRESSION IS DONE SUCCESSFULLY");
        System.out.print("\n\n");
    }
}

```

## **RSA Decryption**

### **NRSA.java**

```

import java.io.*;
import java.util.*;
import java.math.BigInteger;
class NRSA
{
    public static void main(String args[]) throws IOException
    {
        int c;
        String s1="";
        String s2="";
        String s3="";
        DataInputStream ds=new DataInputStream(System.in);
        FileInputStream fin=new FileInputStream("keys.txt");
        FileInputStream fin2=new FileInputStream("decompress.txt");
        FileOutputStream fout=new FileOutputStream("originaloutput.txt");
        while((c=fin.read())!=-1)
        {
            if((char)c==':')
            {
                break;
            }
            else
            {
                s1+=(char)c;
            }
        }
        c=fin.read();
        while((c=fin.read())!=-1)

```



```

{
    s2+=(char)c;
}

BigInteger prvkey=new BigInteger(s1);
BigInteger r=new BigInteger(s2);

while((c=fin2.read())!=-1)
{
    if((char)c!=':')
    {
        s3+=(char)c;
    }
    else if(s3!="")
    {

        BigInteger cival=new BigInteger(""+s3);
        BigInteger plval=cival.modPow(prvkey,r);
        int d=plval.intValue();
        fout.write(d);
        s3="";
    }
}
System.out.println("\n\nDECRYPTED SUCCESSFULLY\n\n");
System.out.println("ORIGINAL FILE RECEIVED\n\n");
}
}

```

## 6. Sample Output

### Step1: Creation of Encrypted file



```
C:\Windows\system32\cmd.exe

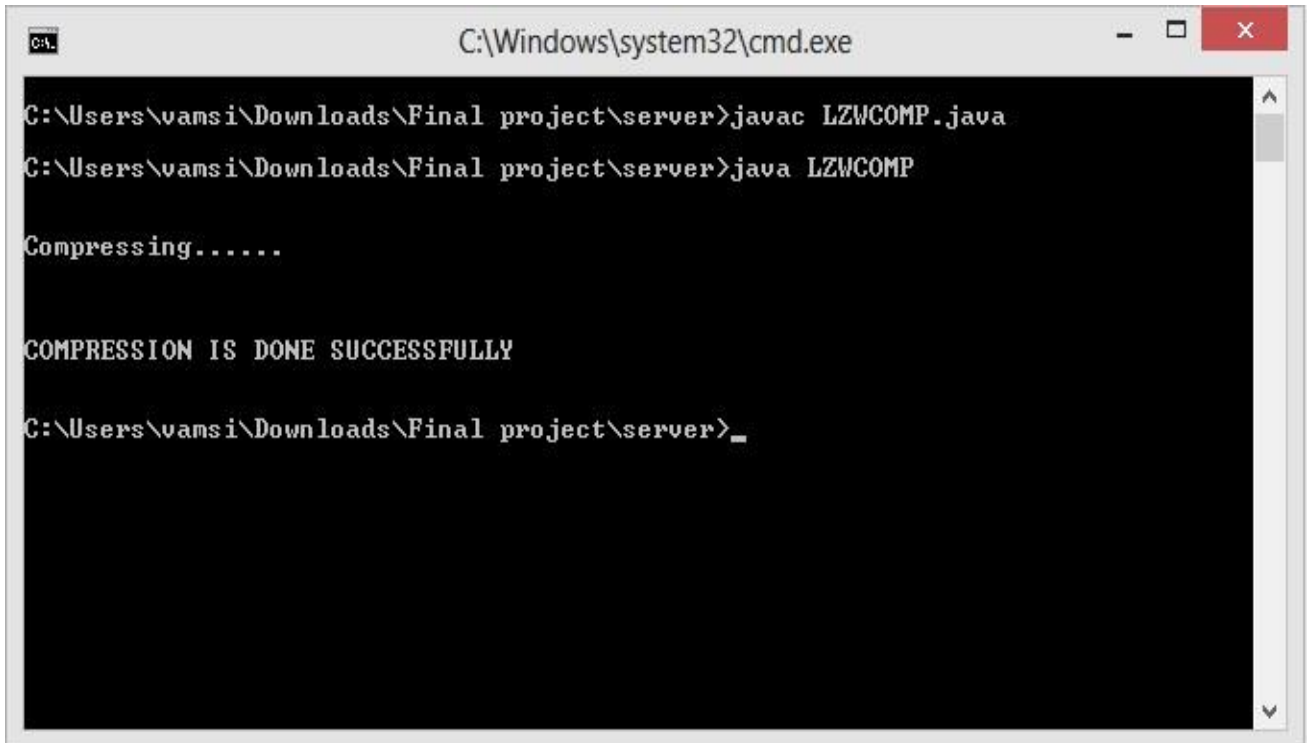
C:\Users\vamsi\Downloads\Final project\server>javac RSA.java
Note: RSA.java uses or overrides a deprecated API.
Note: Recompile with -Xlint:deprecation for details.

C:\Users\vamsi\Downloads\Final project\server>java RSA
Enter some sample public key value:
43

ENCRYPTION IS SUCCESSFULL

C:\Users\vamsi\Downloads\Final project\server>_
```

### Step2: Creation of Compressed file



```
C:\Windows\system32\cmd.exe

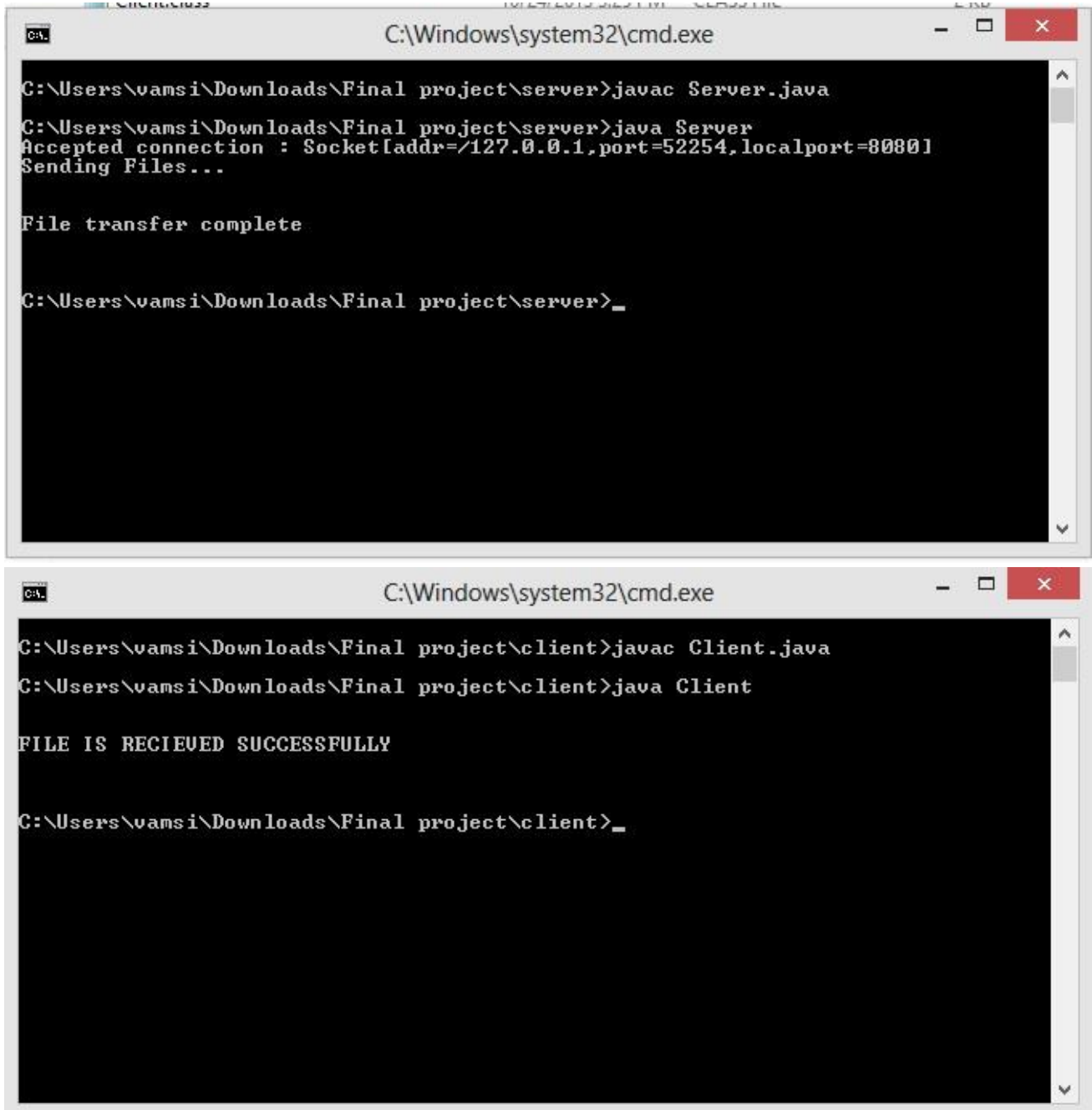
C:\Users\vamsi\Downloads\Final project\server>javac LZWCOMP.java
C:\Users\vamsi\Downloads\Final project\server>java LZWCOMP

Compressing.....

COMPRESSION IS DONE SUCCESSFULLY

C:\Users\vamsi\Downloads\Final project\server>_
```

**Step3:** Sending Compressed and Encrypted file to client system



The image displays two separate Windows command prompt windows, both titled "C:\Windows\system32\cmd.exe".

The top window shows the execution of a Java server program. The commands entered are `javac Server.java` and `java Server`. The output indicates that the server accepted a connection from `127.0.0.1` on port `52254` (local port `8080`) and successfully sent files. The message "File transfer complete" is displayed, followed by a prompt for the next command.

The bottom window shows the execution of a Java client program. The commands entered are `javac Client.java` and `java Client`. The output displays the message "FILE IS RECIEVED SUCCESSFULLY" (note the spelling error in the original image). The prompt then returns to the command line.

```
C:\Windows\system32\cmd.exe
C:\Users\vamsi\Downloads\Final project\server>javac Server.java
C:\Users\vamsi\Downloads\Final project\server>java Server
Accepted connection : Socket[addr=/127.0.0.1,port=52254,localport=8080]
Sending Files...

File transfer complete

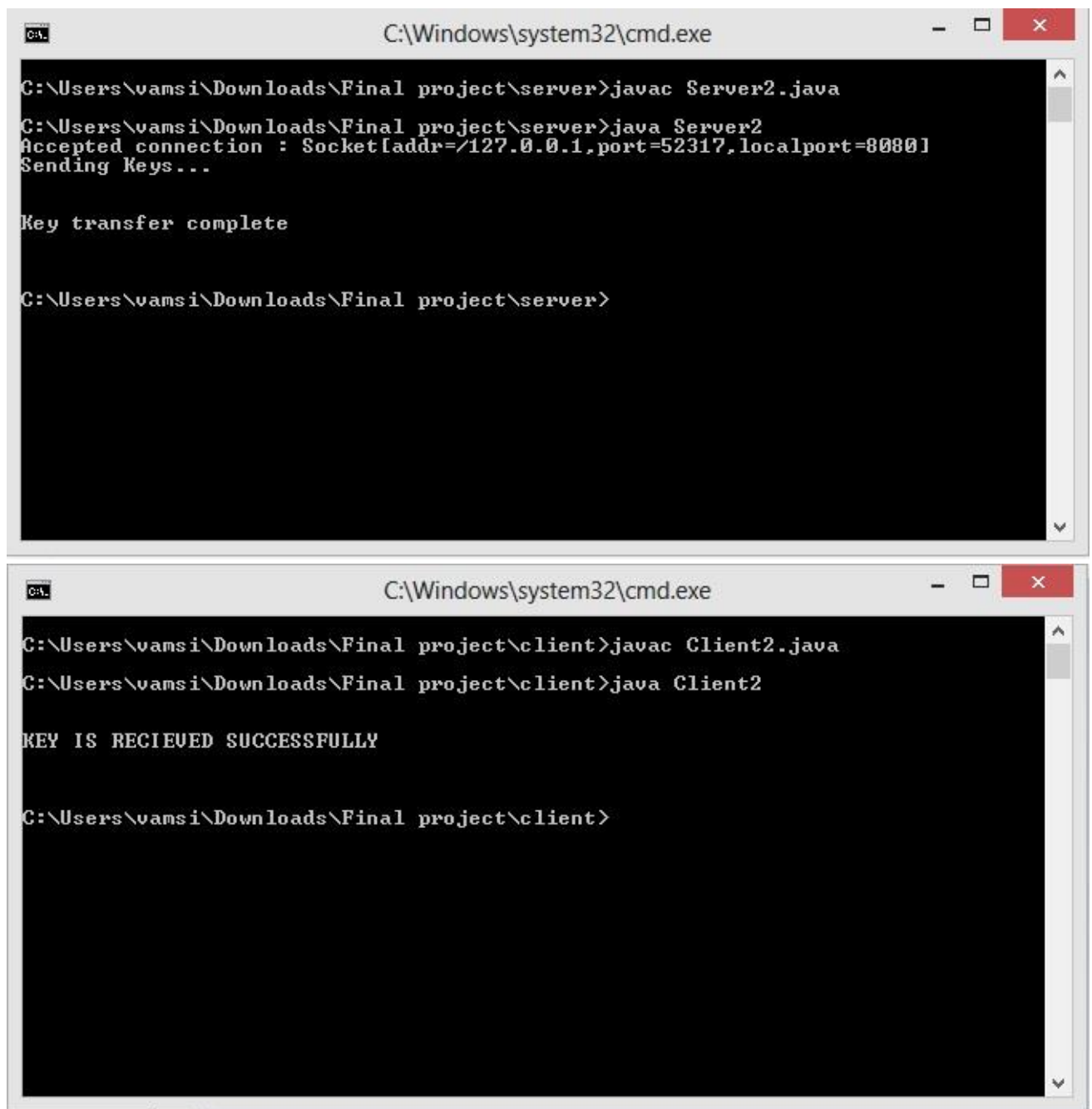
C:\Users\vamsi\Downloads\Final project\server>_

C:\Windows\system32\cmd.exe
C:\Users\vamsi\Downloads\Final project\client>javac Client.java
C:\Users\vamsi\Downloads\Final project\client>java Client

FILE IS RECIEVED SUCCESSFULLY

C:\Users\vamsi\Downloads\Final project\client>_
```

**Step4:** Sending private key to client system



The image displays two separate Windows command prompt windows, both titled "C:\Windows\system32\cmd.exe".

The top window shows the execution of a Java server program. The commands entered are:

```
C:\Users\vamsi\Downloads\Final project\server>javac Server2.java
C:\Users\vamsi\Downloads\Final project\server>java Server2
```

The output of the second command is:

```
Accepted connection : Socket[addr=/127.0.0.1,port=52317,localport=8080]
Sending Keys...

Key transfer complete

C:\Users\vamsi\Downloads\Final project\server>
```

The bottom window shows the execution of a Java client program. The commands entered are:

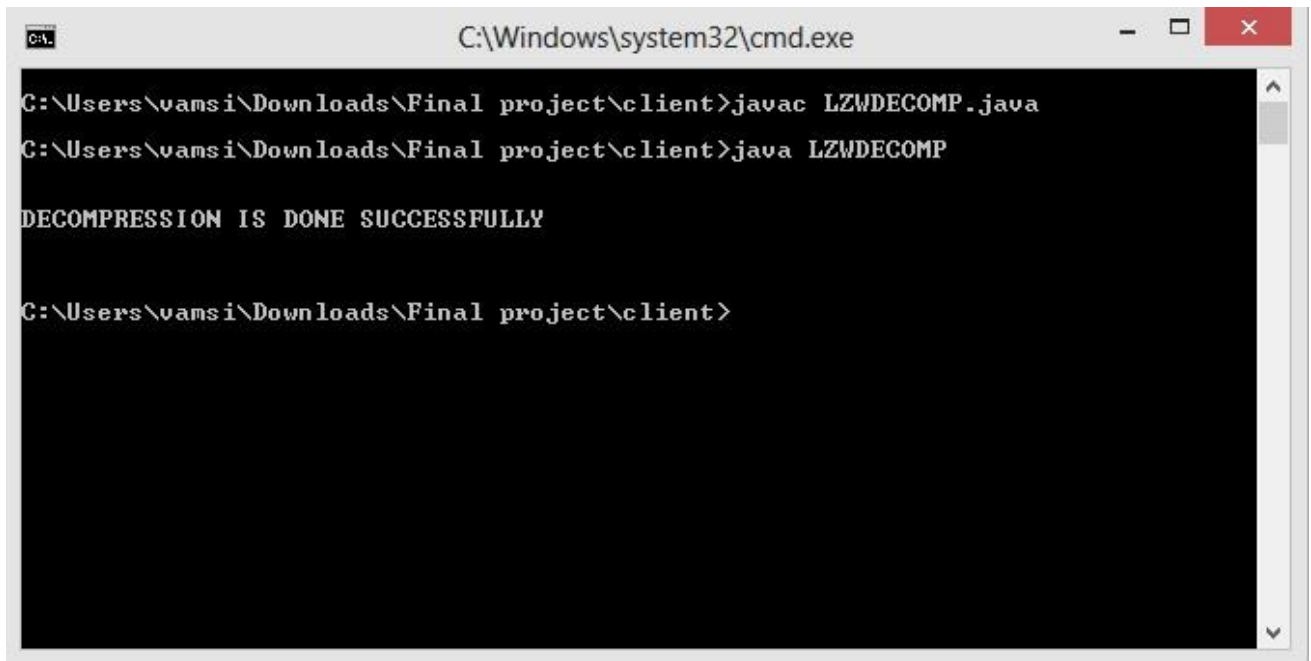
```
C:\Users\vamsi\Downloads\Final project\client>javac Client2.java
C:\Users\vamsi\Downloads\Final project\client>java Client2
```

The output of the second command is:

```
KEY IS RECIEVED SUCCESSFULLY

C:\Users\vamsi\Downloads\Final project\client>
```

**Step5:** Decompressing the Compressed and Encrypted file in client



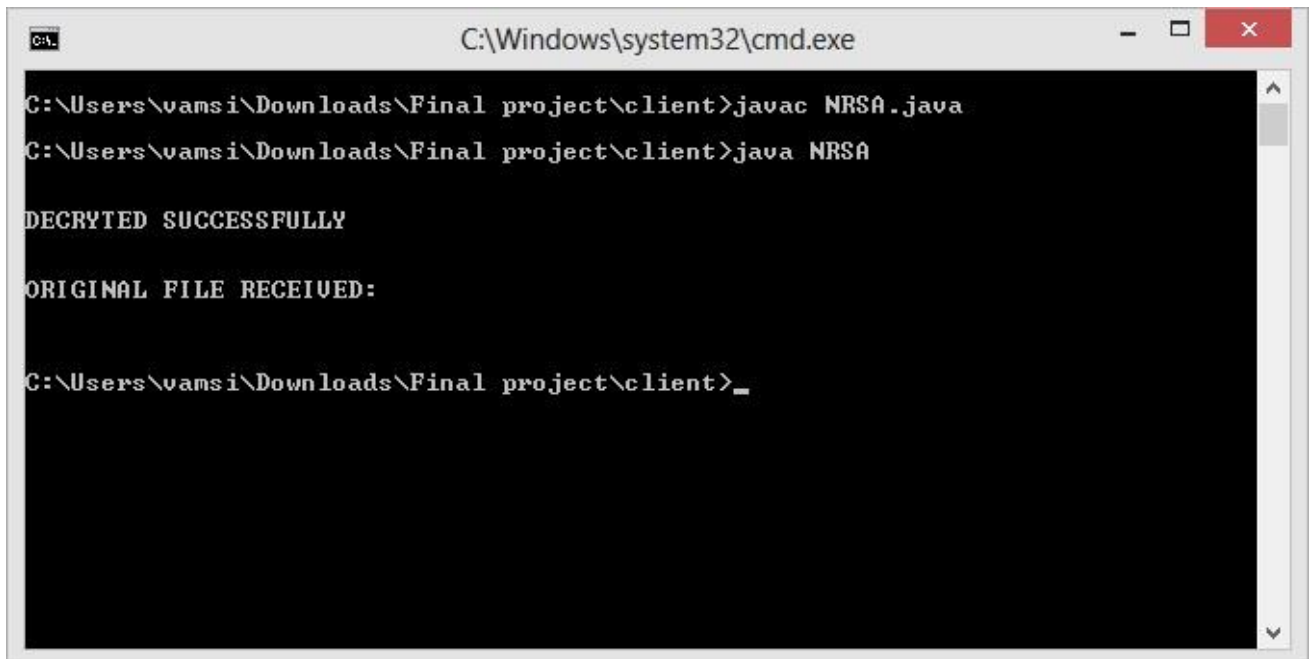
```
C:\Windows\system32\cmd.exe

C:\Users\vamsi\Downloads\Final project\client>javac LZWDECOMP.java
C:\Users\vamsi\Downloads\Final project\client>java LZWDECOMP

DECOMPRESSION IS DONE SUCCESSFULLY

C:\Users\vamsi\Downloads\Final project\client>
```

**Step6:** Decrypting the output of decompression program



```
C:\Windows\system32\cmd.exe

C:\Users\vamsi\Downloads\Final project\client>javac NRSA.java
C:\Users\vamsi\Downloads\Final project\client>java NRSA

DECRYPTED SUCCESSFULLY

ORIGINAL FILE RECEIVED:

C:\Users\vamsi\Downloads\Final project\client>_
```

## 7. Conclusion

While transmitting data from one source to other source two main things are to be concentrated one is security and the other is dependability. In order to provide efficient transfer of data without allowing intruders to guess data, security plays the important role. Dependability is to recover original data in case of any errors. Security of the message is done by providing RSA Encryption algorithm. The message is further encoded by using LZW algorithm which also compress the message. This makes the intruders difficult to guess what the original data is. As the size of the compressed message is reduced faster transmission is possible. Hence by combining RSA and LZW algorithms we can achieve **“Fast and Secure Transmission”**

## **8. References**

- Ebru Celikel Cankaya, Suku Nair, Hakki C. Cankaya, “Applying error correction codes to achieve security and dependability”, *Computer Standards & Interfaces* 35 (2013) 78-86