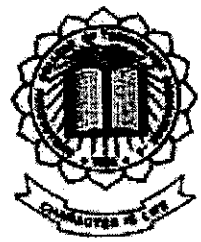


VIRTUAL PRIVATE NETWORKING USING TRIPLE DES ALGORITHM

A PROJECT REPORT

Submitted by

SUJAY.B(71204205054)

VISHNURAAJ.R(71204205059)

in partial fulfillment for the award of the degree

of

BACHELOR OF TECHNOLOGY

in

INFORMATION TECHNOLOGY

KUMARAGURU COLLEGE OF TECHNOLOGY

ANNA UNIVERSITY : CHENNAI 600 025

APRIL 2008

p - 2161

ANNA UNIVERSITY : CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report **"VIRTUAL PRIVATE NETWORKING USING TRIPLE DES ALGORITHM"** is the bonafide work of **SUJAY.B and VISHNURAAJ.R** who carried out the project work under my supervision.

SIGNATURE

Dr.S.Thangasamy

DEAN

Department of
Computer Science and Engineering,
Kumaraguru College Of Technology,
Coimbatore-641006.

SIGNATURE

Ms. N. Chitra Devi M.E.,

SUPERVISOR

Department of Information Technology,
Kumaraguru College Of Technology,
Coimbatore-641006.

The candidates with University Register Nos. **71204205054, 71204205059** examined by us in the project viva-voce examination held on

INTERNAL EXAMINER

EXTERNAL EXAMINER

ACKNOWLEDGEMENT

We express our sincere thanks to our chairman **Padmabhushan Arutselvar Dr. N. Mahalingam B.Sc, F.I.E**, Vice Chairman **Dr. K. Arumugam B.E., M.S., M.I.E.,** Correspondent **Shri.M.Balasubramaniam** and **Joint Correspondent Dr.A Selvakumar** for all their support and ray of strengthening hope extended. We are immensely grateful to our principal **Dr. Joseph V. Thanikal M.E., Ph.D., PDF., CEPIT.,** for his invaluable support to the outcome of this project.

We are deeply obliged to **Dr.S.Thangasamy**, Dean, Department of Computer Science and Engineering for his valuable guidance and useful suggestions during course of this project.

We also extend our heartfelt thanks to our project coordinator **Prof.K.R.Baskaran B.E., M.S.,** Asst. Prof., Department of Information Technology for providing us his support which really helped us.

We are indebted to our project guide **Ms. N. Chitra Devi M.E.,** Sr.Lecturer, Department of Information Technology for her helpful guidance and valuable support given to us throughout this project.

We thank the teaching and non-teaching staffs of our Department for providing us the technical support during the course of this project. We also thank all of our friends who helped us to complete this project successfully.

TABLE OF CONTENTS

ABSTRACT

Virtual Private Network (VPN) allows a connection to a corporate network using a public network (such as the Internet). This technology allows for a low cost connection between corporate Local Area Network and remote users (such as sales representatives or branch locations) regardless of their location.

A VPN is to basically employ a number of security techniques and methods, such as encryption, to allow secure communications.

This project entitled "Virtual Private Networking using triple DES Algorithm" provides a high way security for the file transmission between one location to another location. The file to be transmitted is encrypted at the server side and stored in the buffer. When a valid client connects to the server and request the specified file, it will be transmitted from server to the client.

The client receives the encrypted data from the server and decrypts it.. If the decrypted file is media file, then it provides an option to play it automatically.

This system is grounded on the triple DES technique such that system resilience against attacks is provably strong, given standard computability assumptions. This system provides high network security by triple Encryption and Decryption technique. Keys are securely transmitted using Diffie-Hellman key exchange algorithm.

This system is developed using the language JAVA and JAVA Swing. MS Access is used as the backend. A tool called JMF (Java Media Framework) is used to design the media player.

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

VPN	Virtual Private Network
DES	Data Encryption Standard
JMF	Java Media Framework
LAN	Local Area Network
RSA	Rivest – Shamir – Adleman
GNFS	Generalized Number Field Sieve
SNFS	Specialized Number Field Sieve
API	Application Programming Interface
IP	Inverse Permutation
NIMS	Network Internet Messaging System
ECC	Elliptic Curve Cryptography

INTRODUCTION

GENERAL

1. INTRODUCTION:

1.1. GENERAL:

A "network" has been defined as "any set of interlinking lines or an interconnected system". A computer network is simply a system of interconnected computers

Over the last 25 years or so, a number of networks and network protocols have been defined and used. Anyone can connect to either of these networks, or they can use types of networks to connect their own hosts (computers) together, without connecting to the public networks. Each type takes a very different approach to providing network services.

The Internet is the world's largest network *of* networks . When you want to access the resources offered by the Internet, you don't really connect to the Internet; you connect to a network that is eventually connected to the Internet backbone , a network of extremely fast (and incredibly overloaded) network components.

TCP/IP (Transport Control Protocol/Internet Protocol) is the "language" of the Internet. Anything that can learn to "speak TCP/IP" can play on the Internet. This is functionality that occurs at the Network (IP) and Transport (TCP) layers in the ISO/OSI Reference Model.

A number of attacks against IP like spoofing, session hijacking etc., are possible. Also a number of threats like denial of service, unauthorized access, and confidentiality breaches exist.

Many security measures are available to protect against these attacks. They include cryptographic algorithms, designing firewalls, use of secure network devices like secure modems, crypto – capable routers etc., VPNs provide the

ability for two offices to communicate with each other in such a way that it looks like they're directly connected over a private leased line. Anyone can connect to either of these networks, or they can use types of networks to connect their own hosts (computers) together, without connecting to the public networks.

PROBLEM DEFINITION

1.2. PROBLEM DEFINITION:

The main objective of the project "Virtual Private Networking (VPN) using triple DES algorithm" is to provide security to files that are transmitted from one location to another location. It allows a connection to a corporate network using a public network (such as the Internet). It operates much like a Wide Area Network. It will focus on the common users of VPNs, the security features, and the cost advantages of this type of service.

It is to basically employ a number of security techniques and methods, such as encryption, to allow secure communications to occur between one or more locations via an open network.

Before sending data across the Internet, VPN encrypt the IP data grams using incredibly strong and sophisticated algorithms. At the receiving end, authentication must occur before the intended communication is to be viewed.

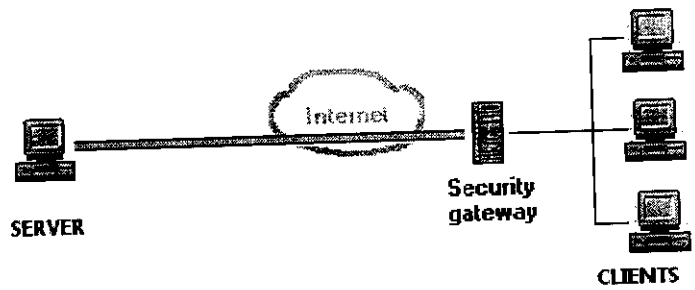


Figure.1

It is a security gateway on the edge of the LAN. It encrypts the data and creates a tunnel to a gateway on the other LAN, where the data is decrypted and placed onto the LAN in its original format.

In this project, implementation of VPN is to provide secured data communication. The encryption of data is done using TRIPLE Data Encryption Standard Algorithm. This is a symmetric algorithm, which uses same keys for both encryption and decryption. The encrypted data is send to another PC through network where the decryption process is carried out.

Encryption for secure transmission in VPN using Data Encryption Standard (DES) is a widely-used method of data encryption using a secret key. **3DES** is a mode of the DES encryption algorithm that encrypts data three times. Three 64-bit keys are used, instead of one, for an overall key length of 192 bits. There are 72,000,000,000,000,000 (72 quadrillion) or more possible encryption keys that can be used.

OBJECTIVE OF THE PROJECT

1.3. OBJECTIVE OF THE PROJECT:

The main objective of the project "Virtual Private Networking (VPN) using triple DES algorithm" is to provide security to files that are transmitted from one location to another location. . The encryption of data is done using Triple Data Encryption Standard Algorithm. This is a symmetric algorithm, which uses same keys for both encryption and decryption.

3DES algorithm provides for secure encryption of the messages. Encryption is carried out using the keys that are securely transferred by Diffie-Hellman Key Exchange. Brute force and timing attacks are practically impossible. It exhibits considerable cryptographic strength and provides backward compatibility.

LITERATURE REVIEW

FEASIBILITY STUDY

CURRENT STATUS OF THE PROBLEM

2. LITERATURE REVIEW:

2.1. FEASIBILITY STUDY:

2.1.1. CURRENT STATUS OF THE PROBLEM:

The existing system makes use of RSA algorithm. Rivest, Shamir and Adleman developed RSA algorithm in 1977.

The RSA scheme is a block cipher in which the plain text and cipher text are integers between 0 and n-1 for some n. A typical size for n is 1024 bits or 309 decimal digits. Both sender and receiver must know the value of n. the sender knows the value of e, and only the receiver knows the value of d. Both encryption and decryption in RSA involve raising an integer to an integer power, mod n. It makes use of an expression with exponentials.

Plain text is encrypted in blocks, with each block having a binary value less than some number n. that is, the block size must be less than or equal to $\log_2(n)$. In practice, the block size is k bits, where $2^k < n \leq {}^{2k+1}$. A corollary to Euler's theorem fits the algorithm.

LIMITATIONS OF THE CURRENT SYSTEM:

Timing attack is possible. Factoring problem also occurs. RSA Algorithm can be easily hacked using factoring algorithms. The three Approaches to attacking RSA mathematically are:

- Factor n into two primes. This enables calculation of $\phi(n)$, which, in turn, enables the calculation of d.
- Determine $\phi(n)$ directly, without first determining p and q. Again, this enables determination of d.
- Determine d directly, without first determining $\phi(n)$.

For a large n with large prime factors, factoring is a hard problem, but not as hard as it is used to be. In 1977, the three inventors of the algorithm challenged that the algorithm cannot be hacked for 40 quadrillion years from the year of its invention. They offered a $100 reward for the return of plain sentence. But in April 1994, a group of Scientific American readers hacked the algorithm after only eight years of work. This challenge used a public key of 129 decimal digits or around 428 bits.

Until the mid 1990s, factoring attacks were made using an approach known as the Quadratic sieve. The attack on RSA-130 used a newer algorithm, the Generalized Number Field Sieve (GNFS), and was able to factor a larger number than RSA-129 at only 20% of the computing work. The following table illustrates the algorithms used to hack RSA algorithm and its details. The level of effort is measured in MIPS-years: a million instructions-per-second processor running for one year.

Table 1

Progress in Factorization

Number of bits	MIPS-years	Factoring Algorithm used
332	7	Quadratic Sieve
365	75	Quadratic Sieve
398	830	Quadratic Sieve
428	5000	Quadratic Sieve
431	1000	GNFS
465	2000	GNFS
512	8000	GNFS

In fact, a related algorithm, the specialized number field sieve (SNFS), can factor numbers with a specialized form considerably faster than the generalized number filed sieve. The following graph compares the performance of the two algorithms.

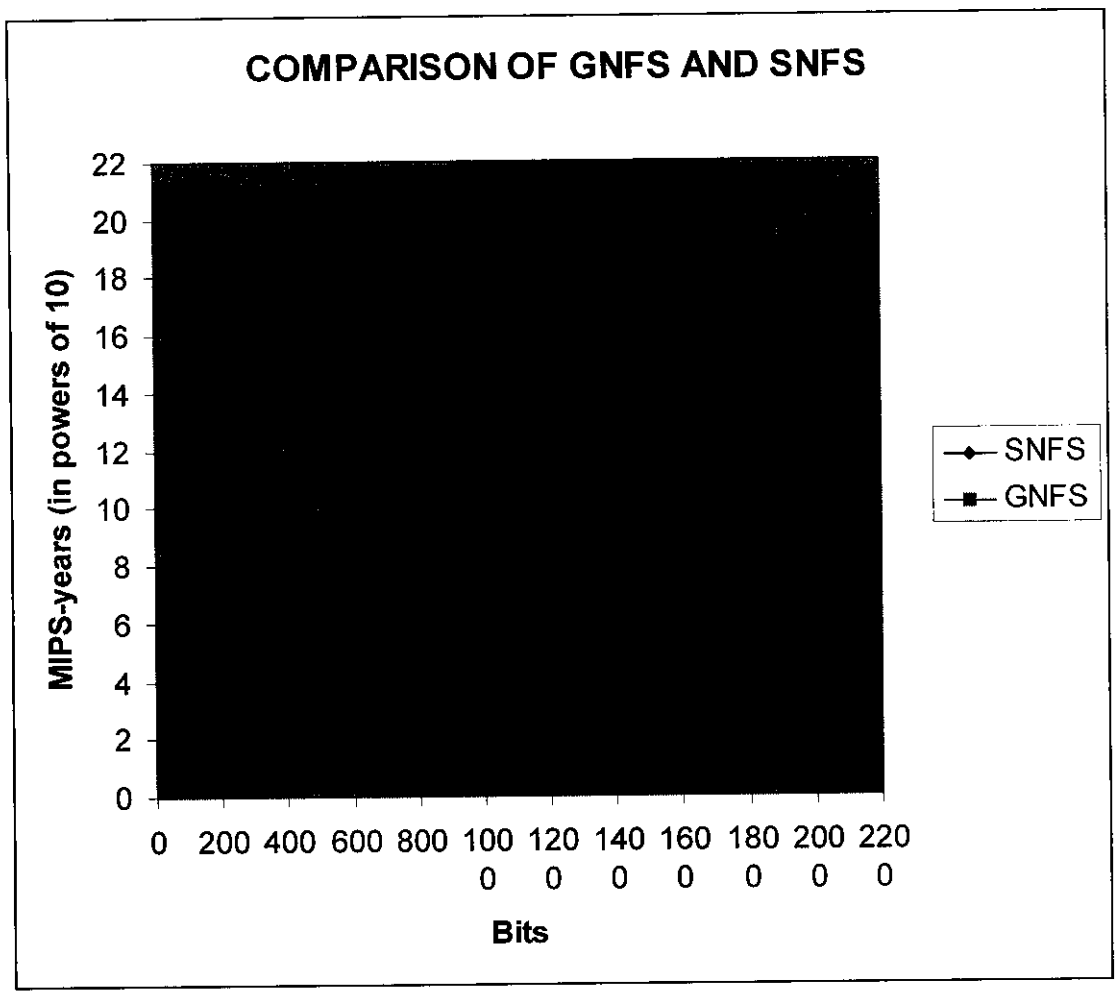

COMPARISON OF GNFS AND SNFS
MIPS-years (in powers of 10)
22
20
18
16
14
12
10
8
6
4
2
0
0
200
400
600
800
1000
1200
1400
1600
1800
2000
2200
Bits
SNFS
GNFS

Figure.2

PROPOSED SYSTEM AND ITS ADVANTAGES

2.1.2. PROPOSED SYSTEM AND ITS ADVANTAGES:

File is securely transferred from server to client using triple DES algorithm

Here three keys are used for encryption and decryption. This algorithm is three times more secure than DES algorithm. Keys are securely transferred using Diffie-Hellman key exchange.

A virtual private network (VPN) is a method of providing security to a local area network over a shared or public infrastructure like the Internet. VPN creates a wide area network or an extranet within the organization, which eliminates the need of installing, dedicated lines for the purpose of communication. It builds up a passage from which the encrypted data is channeled and ultimately reaches the other side and is then decrypted by the receiver

The Data Encryption Standard (DES) used in the proposed system is a widely used method of data encryption using a secret key. **3DES** is a mode of the DES encryption algorithm that encrypts data three times. Three 56-bit keys are used, instead of one, for an overall key length of 168 bits (the first encryption is again decrypted with second key, and the resulting data is again encrypted with a third key).

There are 72,000,000,000,000,000 (72 quadrillion) or more possible encryption keys that can be used. For each given message, the key is chosen at random from among this enormous number of keys.

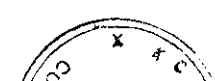
ADVANTAGES OF THE PROPOSED SYSTEM:

1. Brute force attack is impossible. Larger key size - 3 56 bit keys are used which provides 72,000,000,000,000,000 (72 quadrillion) or more possible encryption keys.
2. No timing attack as encryption and decryption done at different times.
3. 3DES exhibit considerable cryptographic strength.
4. Provides backward compatibility.
5. Diffie-Hellman key exchange allows for secure transfer of keys.
6. Encryption with 3DES is a mapping of 64-bit blocks to 64-bit blocks. In fact, the mapping can be viewed as a permutation. That is, if we consider all 2^{64} possible input blocks, DES encryption with a specific key will map each block into a unique 64-bit block. Otherwise, if, say, two given input blocks mapped to the same output block, then decryption to recover the plain text would be impossible. With 264 possible inputs, the number of different mappings are

$$2^{64}!= 10^{34738000000000000000} > (10^{10(20)})$$

Meet-in-the-Middle Attack:

Given a known pair, (P, C), the attack proceeds as follows:
First, encrypt P for all 256 possible values of k_1, Store these results in a table and sort the table by the values(X). Next. Decrypt C using all 256 possible values of k_2. A each decryption is produced, check the two



resulting keys against a new known plaintext – cipher text pair. If the two keys produce the correct cipher text, accept them as the correct keys.

It has been proved that Meet-in-the-Middle Attack is practically impossible on 3DES. "**It is theoretically possible but practically impossible**".

The reason is that encryption and decryption occurs at different times. Even if an intruder guesses the key, it would be of no use because the intended person would have already received the message, as it takes a longer time for brute force attack. Thus, the message becomes invalid when an intruder hacks it.

HARDWARE AND SOFTWARE REQUIREMENTS

2.2. HARDWARE REQUIREMENTS:

Server:

Processor	:	Intel Pentium IV
RAM	:	256Mega Bytes
Hard Disk	:	10 Giga Bytes
Operating System	:	Windows XP

Client:

Processor	:	Intel Pentium IV
RAM	:	128 Mega Bytes
Hard Disk	:	10 Giga Bytes
Operating System	:	Windows XP

2.3. SOFTWARE REQUIREMENTS:

Server:

Front end	:	Java (JDK 1.5)
Back end	:	Microsoft Access

Client:

Front end	:	Java (JDK 1.5)
Tool	:	Java Media Framework

SOFTWARE OVERVIEW

2.4. SOFTWARE OVERVIEW:

JAVA MEDIA FRAMEWORK:

The Java Media Framework (JMF) is a recent API for Java dealing with real-time multimedia presentation and effects processing. JMF handles time-based media, media which changes with respect to time. Examples of this are video from a television source, audio from a raw-audio format file and animations. The beta JMF 2.0 specification will be used for this report, as they currently reflect the features that will appear in the final version.

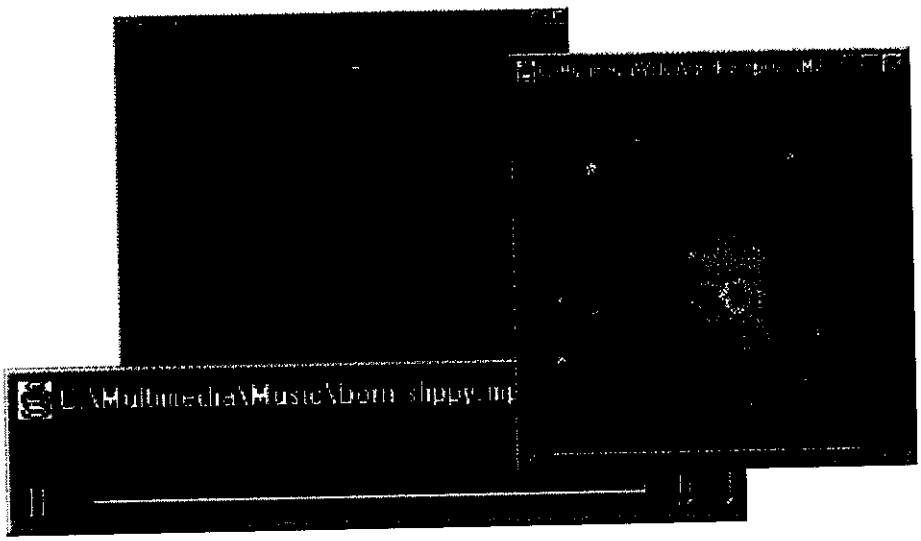


Figure.3

Stages:

The JMF architecture is organized into three stages:

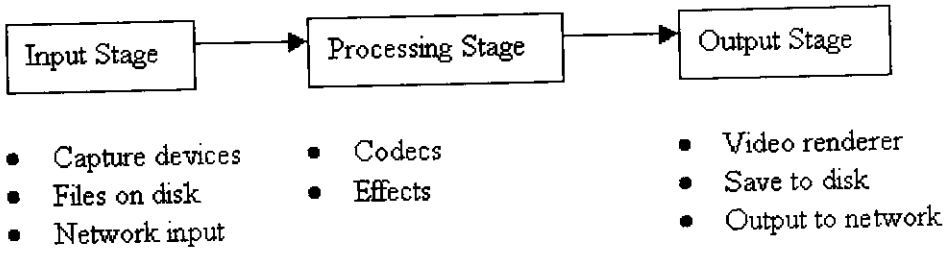


Figure.4

During the input stage, data is read from a source and passed in buffers to the processing stage. The input stage may consist of reading data from a local capture device (such as a webcam or TV capture card), a file on disk or stream from the network.

The processing stage consists of a number of codecs and effects designed to modify the data stream to one suitable for output. These codecs may perform functions such as compressing or decompressing the audio to a different format, adding a watermark of some kind, cleaning up noise or applying an effect to the stream (such as echo to the audio).

Once the processing stage has applied its transformations to the stream, it passes the information to the output stage. The output stage may take the stream and pass it to a file on disk, output it to the local video display or transmit it over the network.

For example, a JMF system may read input from a TV capture card from the local system capturing input from a VCR in the input stage. It may then pass it to the processing stage to add a watermark in the corner of each frame and finally broadcast it over the local Intranet in the output stage.

Component Architecture:

JMF is built around a component architecture. The compenents are organized into a number of main categories:

- Media handlers
- Data sources
- Codecs/Effects
- Renderers
- Mux/Demuxes

Media Handlers

MediaHandlers are registered for each type of file that JMF must be able to handle. To support new file formats, a new MediaHandler can be created.

Data Sources

A DataSource handler manages source streams from various inputs. These can be for network protocols, such as http or ftp, or for simple input from disk.

Codecs/Effects

Codecs and Effects are components that take an input stream, apply a transformation to it and output it. Codecs may have different input and output formats, while Effects are simple transformations of a single input format to an output stream of the same format.

Renderers

A renderer is similar to a Codec, but the final output is somewhere other than another stream. A VideoRenderer outputs the final data to the screen, but another kind of renderer could output to different hardware, such as a TV out card.

Mux/Demuxes

Multiplexers and Demultiplexers are used to combine multiple streams into a single stream or vice-versa, respectively. They are useful for creating and reading a package of audio and video for saving to disk as a single file, or transmitting over a network.

DETAILS OF THE METHODOLOGY EMPLOYED

3. DETAILS OF THE METHODOLOGY EMPLOYED:

DIFFIE-HELLMAN KEY EXCHANGE:

Server side

- ✓ Admin Authentication
- ✓ Listen and key exchange

Client side

- ✓ Connection establishment
- ✓ User Authentication
- ✓ Key Exchange

3DES:

Server side

- ✓ Admin Authentication
- ✓ Encryption

Client side

- ✓ Connection establishment
- ✓ User Authentication
- ✓ Decryption
- ✓ Autoplay

DIFFIE-HELLMAN KEY EXCHANGE:

Admin authentication:

At the Server, the administrator authenticates himself by providing his user name and password. If he is an authorized person, he gets access to the machine. Now the server is ready to listen for any incoming connections. If an unauthorized person tries accessing the server, the system gets shut down.

Listen And Key exchange:

The server listens for any requests for connection from valid clients. If any, it accepts the connection. Once the connection has been established, the server receives the public keys of the corresponding client. In turn, the server calculates the public keys using its private keys and passes them to the client. Using the public keys of the client, the server calculates the encryption keys and stores the corresponding information in its domain. Keys are securely transmitted using Diffie-Hellman key exchange algorithm.

Connection Establishment:

Before any key exchange, the client must establish connection with the server. The user provides the IP address of the server to communicate and the port number. Connection is established between the server and the corresponding client.

Client authentication:

Once the connection has been established with the required server, the client authenticates him by providing his user name and password. If he is an authorized person, the connection is established with the server. If not, the system shuts down. The login details resides on the server.

Table.2

Authentication

User_Name	Pass_Words	prikey1	prikey2
admin	admin	22	43
sujay	04bit55	34	26
vishnu	04bit60	102	24

Key Exchange:

The user enters his private keys. Corresponding public keys are calculated and transmitted to the server. In turn, the client receives the public keys of the server. Using them, the client calculates the decryption keys and stores them in its domain. Keys are securely transferred using Diffie-Hellman key exchange algorithm.

Diffie-Hellman Key Exchange Algorithm

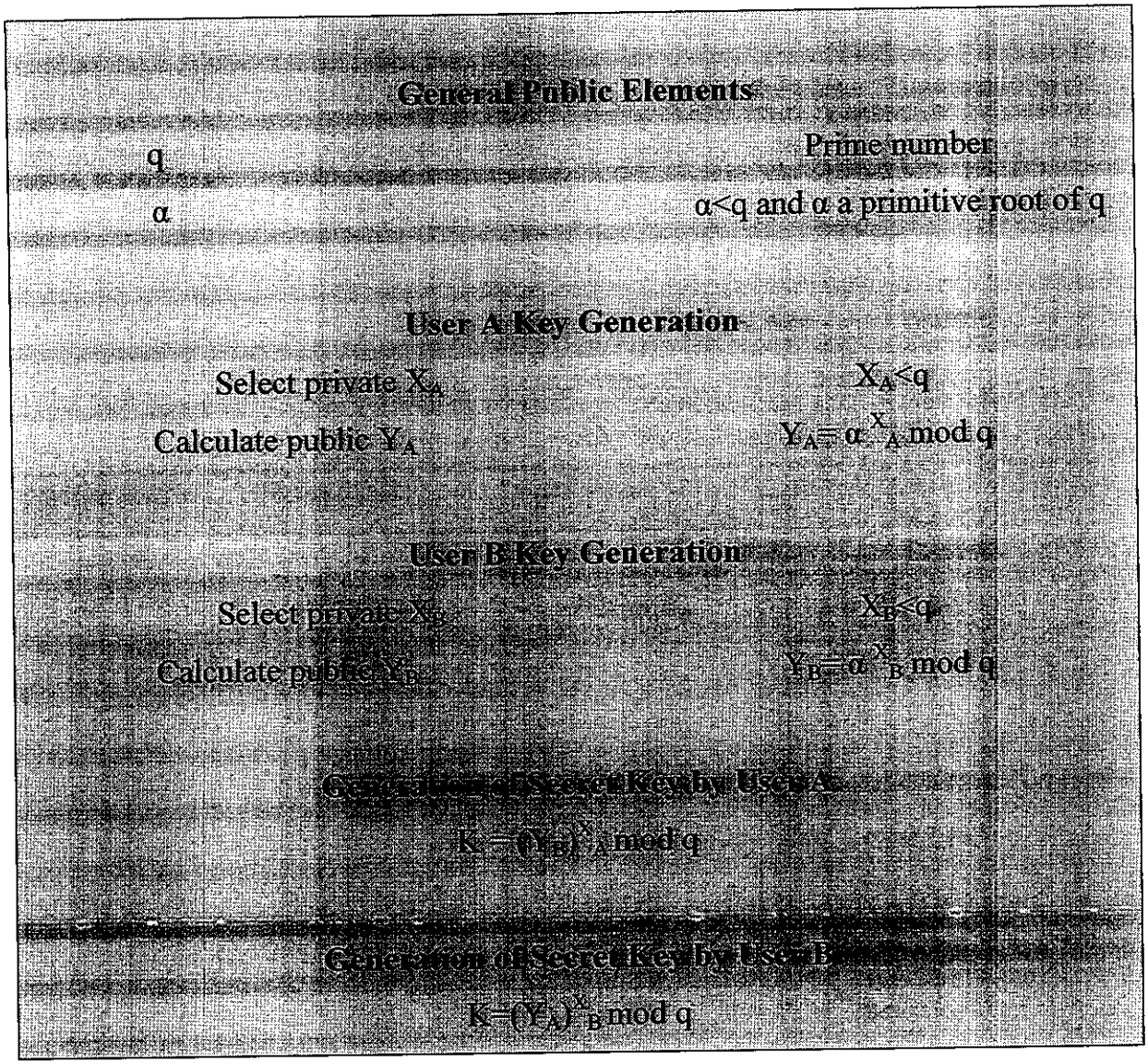


Figure.5

3DES:

ENCRYPTION:

The server enters the file requested from the client along with the corresponding encryption keys stored in its domain. The server encrypts the file using the 3DES algorithm.

Encryption

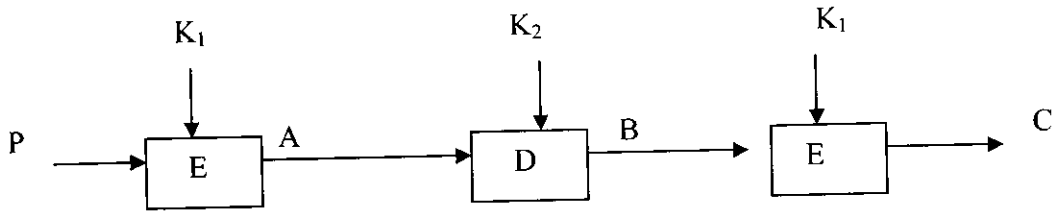


Figure.6

DECRYPTION:

The client specifies the location of the file and enters the decryption keys stored in its domain. Decryption uses the same algorithm as encryption, except that the application of the sub keys is reversed. Once the file is decrypted, it is stored in the specified location.

Decryption

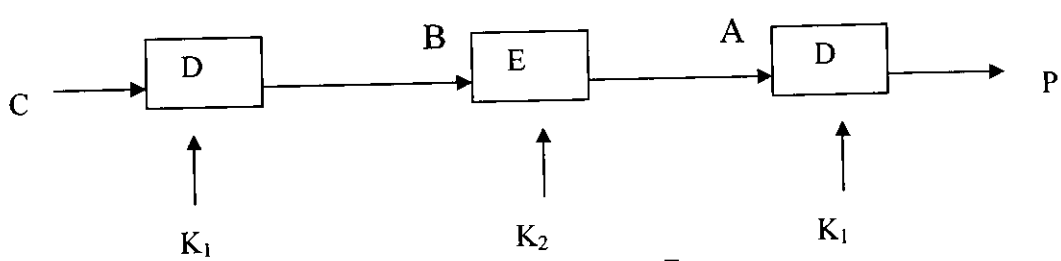


Figure.7

General Description of DES Algorithm:

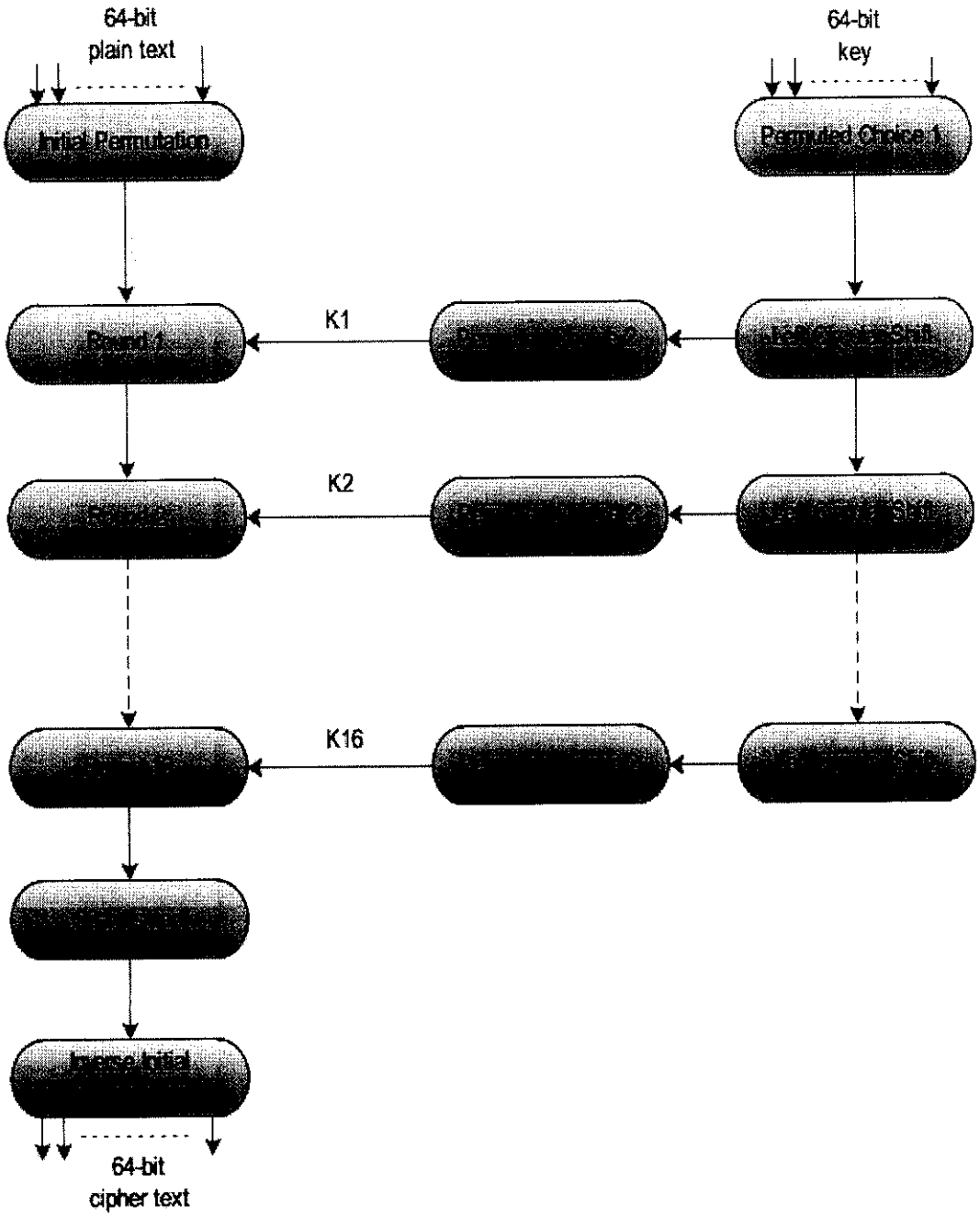


Figure.8

There are two inputs to the encryption function: the plain text to be encrypted and the key. The plain text must be 64 bits in length and the key is 56 bits in length. The processing of the plain text proceeds in three phases. First, the 64-bit plain text passes through an initial permutation (IP) that rearranges the bits to produce the permuted input. This is followed by a phase consisting of 16 rounds of the same functions.

Single round of DES Algorithm

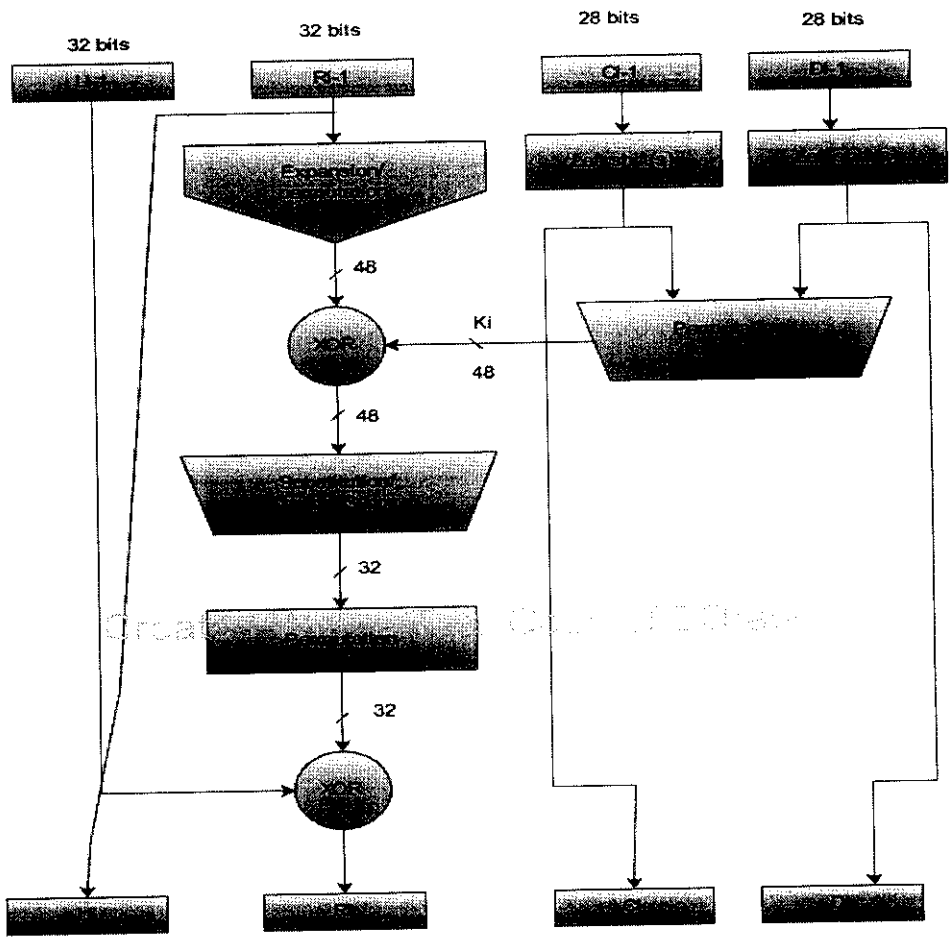


Figure.9

The output of the last round consists of 64 bits that are a function of the input plaintext and the key. The left and right halves of the output are swapped to produce the pre-output. Finally, the pre-output is passed through a permutation (IP^{-1}) that is the inverse of the initial permutation function., to produce the 64-bit cipher text. With the exception of the initial and final permutations, 3DES has the exact structure of a Feistel cipher.

Autoplay:

The system provides an option to play audio or video files automatically using Java Media Framework (JMF). When the client decrypts the media file, the system provides the user with this option.

CONCLUSION

4. CONCLUSION:

Thus the project "VIRTUAL PRIVATE NETWORKING USING TRIPLE DES " has successfully developed and implemented on the concept of Triple Data Encryption Standard (DES) .It provides security for the data transmitted between systems.

It is a good choice for military use where keys can be exchanged securely using Diffie-Hellman key exchange.

It can also be used to encrypt data streams over private network routers and to encrypt data stored on disk.

Thus Virtual Private Networking allows for a low cost connection between corporate Local Area Network (LAN) and remote users regardless of location.

FUTURE ENHANCEMENTS

5. FUTURE ENHANCEMENTS:

The project can be developed into a completely secure and portable system starting from Encryption, Stenography, compression to image enhancement-restoration, decompression, destenography and decryption.

In future it can be enhanced for Net Management Services which is provided for Web based management of applications such as eManager, Group wise and NIMS (Network Internet Messaging System), eDirectory and Netware.

Using Virtual Private Networking, these management tools increase security without decreasing management efforts.

Elliptic Curve Cryptography (ECC) can be employed for the secure exchange of keys. Diffie-Hellman key exchange algorithm is also secure but as the key length increases, the time increases along with the increased efficiency. The additional time is negligible. But in elliptic curve cryptography, increased efficiency can be obtained without any change in time. The efficiency is achieved as Elliptic Curve Cryptography uses only shorter keys, which provide high security.

In this method, the first task is to encode the plain text message m to be sent a an x – y point pm. It is the point pm that will be encrypted as a cipher text and subsequently decrypted. The message cannot be simple encoded as the x or y coordinate of a point, because not all such coordinates are in $E_q(a,b)$. as with the key exchange system, an encryption/decryption system requires a point G and an elliptic group $E_q(a,b)$ as parameters. Each user A selects a private key n_A and generates a public key $P_A = n_A$ X G. To encrpyt and send a message Pm to B, A chooses a random positive integer k and produces the cipher text Cm

consisting of the pair of points

$C_m = \{kG, P_m + kP_B\}$

Note that A has used B's public key PB. To decrypt the cipher text, B multiplies the first point in the pair by B's secret key and subtracts the result from the second point:

$P_m + kP_B - n_B(kG) = P_m + k(n_BG) - n_B(kG) = P_m$

A has masked the message Pm by adding kP_B to it. Nobody but A knows the value of k, so even though P_B is a public key, nobody can remove the mask of P_B. However, A also includes a "clue", which is enough to remove the mask if one knows the private key n_B. For an attacker to recover the message, the attacker would have to compute k given G and kG, which is assumed hard.

Digital signatures can also be employed for more security. The digital signature provides a set of security capabilities that would be difficult to implement in any other way. It involves either the communicating parties (directed) or an arbiter (arbitrated). A digital signature may be formed by encrypting the entire message with the sender's private key or by encrypting a hash code of the message with the sender's private key. Confidentiality is provided by further encrypting the entire message plus signature with the receiver's public key(public-key encryption) or a shared symmetric key(symmetric encryption). A centralized trusted agent can be used for the exchange of keys.

3DES can be used together with El Gamel encryption algorithm which is the combination of digital signature and ECC.

APPENDICES

6.APPENDICES:

APPENDIX 1:

SAMPLE CODE:

Authentication:

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.sql.*;
import java.util.*;
import java.text.*;

public class Server_Logon extends JFrame implements ActionListener
{
        private Dimension d = Toolkit.getDefaultToolkit().getScreenSize(); //Getting the
User's Screen Dimensions.
        private JPanel pLog = new JPanel();
        private JLabel lbUser, lbPass;
        private JTextField txtUser;
        private JPasswordField txtPass;
        private JButton btnOk, btnCancel;
        private Connection con;                 //For Creating the Connection Between Program
& Database.
        public String user;             //For Getting the Current User's Name
        public Server_Logon()
        {

          //Setting Program's Title.

                super ("TDES.");

                //Setting the Main Window of Program.

                setIconImage (getToolkit().getImage ("Logo.gif")); //Setting the Program's
                                                                        Icon.
                setSize (275, 150);     //Setting Main Window Size.
                setResizable (false);   //Make it UnResizeable.

                //Closing Code of Main Window.
                addWindowListener (new WindowAdapter ()
```

```
        {       //Attaching the WindowListener to Program.

                public void windowClosing (WindowEvent we)
                {       //Overriding the windowClosing Function.

                  setVisible (false);                   //Hide the Form.
                  dispose();                            //Free the System Resources.
                  System.exit (0);              //Close the Application.
                }
        }
);

//Setting the Logon Form Position on User's Screen.

setLocation (d.width / 2 - getWidth() / 2, d.height / 2 - getHeight() / 2);

//Setting the Layout of Panel.

pLog.setLayout (null);

//Setting the Form's Labels.

lbUser = new JLabel ("Username:");
lbUser.setForeground (Color.black);
lbUser.setBounds (20, 15, 75, 25);
lbPass = new JLabel ("Password:");
lbPass.setForeground (Color.black);
lbPass.setBounds (20, 50, 75, 25);

//Setting the Form's TextField & PasswordField.

txtUser = new JTextField ();
txtUser.setBounds (100, 15, 150, 25);
txtPass = new JPasswordField ();
txtPass.setBounds (100, 50, 150, 25);

//Setting the Form's Buttons.

btnOk = new JButton ("OK");
btnOk.setBounds (20, 90, 100, 25);
btnOk.addActionListener (this);
btnCancel = new JButton ("Cancel");
btnCancel.setBounds (150, 90, 100, 25);
btnCancel.addActionListener (this);

//Adding All the Controls in Panel.
```

```
        pLog.add (lbUser);
        pLog.add (lbPass);
        pLog.add (txtUser);
        pLog.add (txtPass);
        pLog.add (btnOk);
        pLog.add (btnCancel);

        //Adding Panel to the Form.

        getContentPane().add (pLog);

        //Showing The Logon Form.

        setVisible (true);

}

public void actionPerformed (ActionEvent ae)
{

  Object obj = ae.getSource();

  if (obj == btnOk)
  {             //If OK Button Pressed.

        String password = new String (txtPass.getPassword());
        if (txtUser.getText().equals (""))
        {
              JOptionPane.showMessageDialog (this, "Provide Username to Logon.");
              txtUser.requestFocus();
        }
        else if (password.equals (""))
        {
              txtPass.requestFocus();
              JOptionPane.showMessageDialog (null,"Provide Password to Logon.");
        }
        else
        {
              String pass;                        //To Hold the Password.
              boolean verify = false;                   //To Confirm Logon.
             //SELECT Query to Retrieved the Record.
              user = "admin";                     //Storing UserName.
              pass = "admin";                     //Storing Password.

              if (txtUser.getText().equals (user) && password.equals (pass))
              {//If Found then.
```

```
                verify = true;
                new StartServer();    //Show Main Form.
                setVisible (false);           //Hide the Form.
                dispose();                    //Free the System Resources.
               }
               else
               {
                verify = false;
                JOptionPane.showMessageDialog (this, "Incorrect Information Provided.");
                txtUser.setText ("");
                txtPass.setText ("");
                txtUser.requestFocus ();
               }

        }

       }
       else if (obj == btnCancel)
       {              //If Cancel Button Pressed Unload the From.

               setVisible (false);
               dispose();
               System.exit (0);

       }

}

public static void main(String[] args)
{
       new Server_Logon();
}

}
```

Server Key Exchange:

```
import pack2.*;
import java.io.*;
import java.net.*;
import java.util.*;
import java.math.*;
import java.security.Key;
import java.sql.*;
import javax.swing.JOptionPane;
```

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.filechooser.*;
class exchange  implements Runnable

{
  BufferedReader br;
  BufferedOutputStream bo;
  PrintWriter pr;
  InputStream is;
  ServerSocket ss;
  Socket s;
  int size;
  String key1,key2,key3;
  String fname;
  boolean state=true,verify=false;
  double q=353,alpha=3;
  String user = "",Pass = "";
  JScrollPane sc;
  JTextArea ta;
  double xa;
  JTextField t;
  JButton b1,b2;
  JPanel p;
  String str="";
  javax.swing.JFileChooser fc;
  JLabel lbl1,lbl2,lbl3;
  JTextField txt1,txt2,txt3;
  exchange(Socket sock)
  {
   try
        {
                s=sock;
                br=new BufferedReader(new InputStreamReader(sock.getInputStream()));
                pr=new PrintWriter(s.getOutputStream(),true);
                new Thread(this).start();
        }
        catch(Exception e)
        {
          System.out.println("Error "+e);
        }
  }
```

```
public void run()
{
 try
 {
   user = br.readLine();
   Pass = br.readLine();
   try
   {
     //creating object for database connectivity class
     database_conn objDatabase=new database_conn();
     ResultSet rs=objDatabase.stat.executeQuery("select * from
                                                    User_Details");
     while(rs.next())
     {
            String Username=rs.getString(1).trim();
            String Password=rs.getString(2).trim();
            if(Username.equals(user) && Password.equals(Pass))
            {
                  JOptionPane.showMessageDialog(null,"Incoming connection
                              accepted from "+ Username+"....","Information",1);
                  String prikey1=rs.getString(3).trim();
                  String prikey2=rs.getString(4).trim();
                  pr.println("valid");
                  verify=true;
                  String str=br.readLine();
                  JOptionPane.showMessageDialog(null,"Public Key of Client
                                    received Successfully","Information",1);
                  FileOutputStream fos1= new FileOutputStream("E:/Final
                              Project/Source  Code/DES/Server/"+Username+".txt");
                 byte kb1[]=str.getBytes();
                 fos1.write(kb1);
                 double xa1=Double.parseDouble(prikey1);
                 double ya1=Math.pow(alpha,xa1)%q;
                 double xa2=Double.parseDouble(prikey2);
                 double ya2=Math.pow(alpha,xa2)%q;
                 JOptionPane.showMessageDialog(null,"Sending Server's public
                                                    key","Information",1);
                 pr.println("Public Key1:"+ya1+"  Public Key2:"+ya2);
                 break;

            }

           }
           if(!verify)
           {
             pr.println("Invalid");
```

```
                }
                rs.close();
            }
            catch(Exception e)
            {
                System.out.println("Error in fser.java in db "+e );
            }

        }
        catch(Exception e)
        {
          System.out.println(e);
        }
      }
}
```

Client KeyExchange:

```
import pack2.*;
import java.io.*;
import java.net.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.filechooser.*;

public class client extends JInternalFrame implements ActionListener
{
        Socket s;
        PrintWriter pr;
        BufferedReader br;
        ByteArrayOutputStream bos;
        FileOutputStream fout,fos1;
        JTextField tfs,tpk1,tpk2;
        JLabel bpk1,bpk2,bfi;
        JPasswordField tkey1,tkey2,tkey3;
        JTextArea ta;
        JScrollPane sc;
        JButton bs,be,bsc,bde,bs1;
        JPanel p;
        String key1,key2,key3;
        String user;
        javax.swing.JFileChooser fc;
        JDesktopPane desktop;
        String host=null,port=null,Username=null,Password=null,filename="";
```

```
String butDialog[]={"Play","Exit"};
 double q=353,alpha=3;

 public client(String uname,JDesktopPane desk)
 {
  super("Client Access Program",true,true,true,true);
  try
  {

          s = Login.s;
          desktop=desk;
          fc = new JFileChooser();
          bs=new JButton("Send Request");
          bs.setMnemonic(KeyEvent.VK_S);
          be=new JButton("Exit");
          be.setMnemonic(KeyEvent.VK_X);
          bfi=new JLabel("File");
          tfs=new JTextField(17);
          bpk1=new JLabel("Private key1");
          tpk1=new JTextField(17);
          bpk2=new JLabel("Private key2");
          tpk2=new JTextField(17);
          ta=new JTextArea(13,34);
          sc=new JScrollPane(ta);
          p=new JPanel();
          p.setSize(400,250);
          p.setLayout(new GridLayout(8,2));
          p.add(bfi);
          p.add(tfs);
          p.add(bpk1);
          p.add(tpk1);
          p.add(bpk2);
          p.add(tpk2);
          p.add(bs);
          p.add(be);
          getContentPane().add(p);
          getContentPane().add(sc);
          getContentPane().setLayout(new FlowLayout());
          setSize(new Dimension(400,470));
          bs.addActionListener(this);
          be.addActionListener(this);
          setVisible(true);
     }
     catch(Exception e)
     {
                    System.out.println("Error1 "+e);
```

```
  }
}

public void actionPerformed(ActionEvent e)
{
        if(e.getSource()==bs)
        {
         try
         {
                pr=new PrintWriter(s.getOutputStream(),true);
                br=new BufferedReader(new InputStreamReader(s.getInputStream()));
                double xb1=Double.parseDouble(tpk1.getText());
                double yb1=Math.pow(alpha,xb1)%q;
                double xb2=Double.parseDouble(tpk2.getText());
                double yb2=Math.pow(alpha,xb2)%q;
                String fn=tfs.getText();
                String data="File:"+fn+"  Public key1:"+yb1+"  Public key2:"+yb2;

               pr.println(data);
               String pkk=br.readLine();
               FileOutputStream fos1= new FileOutputStream("E:/Final
                                Project/Source Code/DES/Client/keyexchange.txt");

               byte kb1[]=pkk.getBytes();
               fos1.write(kb1);
               ta.setText(pkk);
               JOptionPane.showMessageDialog(null,"Public Key of Server received
                        Successfully","Information",1);
          }
          catch(Exception ae)
          {
                System.out.println("error occured at bs"+ae);
          }
         }

         if(e.getSource()==be)
         {
          try
          {
           this.setClosed(true);
          }
          catch(java.beans.PropertyVetoException pe)
          {
                System.out.println("Error 2"+pe);
          }
         }
```

```
    }
}
```

Encryption:

```
import java.io.*;
public class CipherDES
{
        public static byte[] cryptDES(byte[] input,String key,String mode) throws Exception
        {
                byte[] theKey = key.getBytes();
                byte[][] subKeys = getSubkeys(theKey);

                ByteArrayInputStream bin=new ByteArrayInputStream(input);
                ByteArrayOutputStream bout=new ByteArrayOutputStream();
                byte[] c=new byte[8];

                while(  bin.read(c) !=-1 )
                {
                        //System.out.println("Before size "+c.length);
                        byte[] d=cipher(c,subKeys,mode);
                        bout.write(d);
                        c=null;
                        c=new byte[8];
                }
                return bout.toByteArray();
        }
    static final int[] IP = {
    58, 50, 42, 34, 26, 18, 10, 2,
    60, 52, 44, 36, 28, 20, 12, 4,
    62, 54, 46, 38, 30, 22, 14, 6,
    64, 56, 48, 40, 32, 24, 16, 8,
    57, 49, 41, 33, 25, 17,  9, 1,
    59, 51, 43, 35, 27, 19, 11, 3,
    61, 53, 45, 37, 29, 21, 13, 5,
    63, 55, 47, 39, 31, 23, 15, 7
  };
  static final int[] E = {
    32,  1,  2,  3,  4,  5,
     4,  5,  6,  7,  8,  9,
     8,  9, 10, 11, 12, 13,
    12, 13, 14, 15, 16, 17,
    16, 17, 18, 19, 20, 21,
    20, 21, 22, 23, 24, 25,
    24, 25, 26, 27, 28, 29,
```

```
        28, 29, 30, 31, 32,  1
};
static final int[] P = {
  16,  7, 20, 21,
  29, 12, 28, 17,
   1, 15, 23, 26,
   5, 18, 31, 10,
   2,  8, 24, 14,
  32, 27,  3,  9,
  19, 13, 30,  6,
  22, 11,  4, 25
};
static final int[] INVP = {
  40, 8, 48, 16, 56, 24, 64, 32,
  39, 7, 47, 15, 55, 23, 63, 31,
  38, 6, 46, 14, 54, 22, 62, 30,
  37, 5, 45, 13, 53, 21, 61, 29,
  36, 4, 44, 12, 52, 20, 60, 28,
  35, 3, 43, 11, 51, 19, 59, 27,
  34, 2, 42, 10, 50, 18, 58, 26,
  33, 1, 41,  9, 49, 17, 57, 25
};
private static byte[] cipher(byte[] theMsg, byte[][] subKeys,
  String mode) throws Exception {
  if (theMsg.length<8)
    throw new Exception("Message is less than 64 bits.");
  theMsg = selectBits(theMsg,IP); // Initial Permutation
  int blockSize = IP.length;
  byte[] l = selectBits(theMsg,0,blockSize/2);
  byte[] r = selectBits(theMsg,blockSize/2,blockSize/2);
  int numOfSubKeys = subKeys.length;
  for (int k=0; k<numOfSubKeys; k++) {
      byte[] rBackup = r;
    r = selectBits(r,E); // Expansion
    if (mode.equalsIgnoreCase("encrypt"))
      r = doXORBytes(r,subKeys[k]); // XOR with the sub key
    else
      r = doXORBytes(r,subKeys[numOfSubKeys-k-1]);
    r = substitution6x4(r); // Substitution
    r = selectBits(r,P); // Permutation
    r = doXORBytes(l,r); // XOR with the previous left half
    l = rBackup; // Taking the previous right half
  }
  byte[] lr = concatenateBits(r,blockSize/2,l,blockSize/2);
  lr = selectBits(lr,INVP); // Inverse Permutation
  return lr;
```

```
}

  private static byte[] doXORBytes(byte[] a, byte[] b) {
    byte[] out = new byte[a.length];
    for (int i=0; i<a.length; i++) {
      out[i] = (byte) (a[i] ^ b[i]);
    }
    return out;
  }
  static final int[] S = {
14,  4, 13,  1,  2, 15, 11,  8,  3, 10,  6, 12,  5,  9,  0,  7, // S1
 0, 15,  7,  4, 14,  2, 13,  1, 10,  6, 12, 11,  9,  5,  3,  8,
 4,  1, 14,  8, 13,  6,  2, 11, 15, 12,  9,  7,  3, 10,  5,  0,
15, 12,  8,  2,  4,  9,  1,  7,  5, 11,  3, 14, 10,  0,  6, 13,
15,  1,  8, 14,  6, 11,  3,  4,  9,  7,  2, 13, 12,  0,  5, 10, // S2
 3, 13,  4,  7, 15,  2,  8, 14, 12,  0,  1, 10,  6,  9, 11,  5,
 0, 14,  7, 11, 10,  4, 13,  1,  5,  8, 12,  6,  9,  3,  2, 15,
13,  8, 10,  1,  3, 15,  4,  2, 11,  6,  7, 12,  0,  5, 14,  9,
10,  0,  9, 14,  6,  3, 15,  5,  1, 13, 12,  7, 11,  4,  2,  8, // S3
13,  7,  0,  9,  3,  4,  6, 10,  2,  8,  5, 14, 12, 11, 15,  1,
13,  6,  4,  9,  8, 15,  3,  0, 11,  1,  2, 12,  5, 10, 14,  7,
 1, 10, 13,  0,  6,  9,  8,  7,  4, 15, 14,  3, 11,  5,  2, 12,
 7, 13, 14,  3,  0,  6,  9, 10,  1,  2,  8,  5, 11, 12,  4, 15, // S4
13,  8, 11,  5,  6, 15,  0,  3,  4,  7,  2, 12,  1, 10, 14,  9,
10,  6,  9,  0, 12, 11,  7, 13, 15,  1,  3, 14,  5,  2,  8,  4,
 3, 15,  0,  6, 10,  1, 13,  8,  9,  4,  5, 11, 12,  7,  2, 14,
 2, 12,  4,  1,  7, 10, 11,  6,  8,  5,  3, 15, 13,  0, 14,  9, // S5
14, 11,  2, 12,  4,  7, 13,  1,  5,  0, 15, 10,  3,  9,  8,  6,
 4,  2,  1, 11, 10, 13,  7,  8, 15,  9, 12,  5,  6,  3,  0, 14,
11,  8, 12,  7,  1, 14,  2, 13,  6, 15,  0,  9, 10,  4,  5,  3,
12,  1, 10, 15,  9,  2,  6,  8,  0, 13,  3,  4, 14,  7,  5, 11, // S6
10, 15,  4,  2,  7, 12,  9,  5,  6,  1, 13, 14,  0, 11,  3,  8,
 9, 14, 15,  5,  2,  8, 12,  3,  7,  0,  4, 10,  1, 13, 11,  6,
 4,  3,  2, 12,  9,  5, 15, 10, 11, 14,  1,  7,  6,  0,  8, 13,
 4, 11,  2, 14, 15,  0,  8, 13,  3, 12,  9,  7,  5, 10,  6,  1, // S7
13,  0, 11,  7,  4,  9,  1, 10, 14,  3,  5, 12,  2, 15,  8,  6,
 1,  4, 11, 13, 12,  3,  7, 14, 10, 15,  6,  8,  0,  5,  9,  2,
 6, 11, 13,  8,  1,  4, 10,  7,  9,  5,  0, 15, 14,  2,  3, 12,
13,  2,  8,  4,  6, 15, 11,  1, 10,  9,  3, 14,  5,  0, 12,  7, // S8
 1, 15, 13,  8, 10,  3,  7,  4, 12,  5,  6, 11,  0, 14,  9,  2,
 7, 11,  4,  1,  9, 12, 14,  2,  0,  6, 10, 13, 15,  3,  5,  8,
 2,  1, 14,  7,  4, 10,  8, 13, 15, 12,  9,  0,  3,  5,  6, 11
    };
    private static byte[] substitution6x4(byte[] in) {
      in = splitBytes(in,6); // Splitting byte[] into 6-bit blocks
      byte[] out = new byte[in.length/2];
```

```
 int lhByte = 0;
 for (int b=0; b<in.length; b++) { // Should be sub-blocks
   byte valByte = in[b];
   int r = 2*(valByte>>7&0x0001)+(valByte>>2&0x0001); // 1 and 6
   int c = valByte>>3&0x000F; // Middle 4 bits
   int hByte = S[64*b+16*r+c]; // 4 bits (half byte) output
   if (b%2==0) lhByte = hByte; // Left half byte
   else out[b/2] = (byte) (16*lhByte + hByte);
 }
 return out;
}
private static byte[] splitBytes(byte[] in, int len) {
  int numOfBytes = (8*in.length-1)/len + 1;
  byte[] out = new byte[numOfBytes];
  for (int i=0; i<numOfBytes; i++) {
    for (int j=0; j<len; j++) {
      int val = getBit(in, len*i+j);
      setBit(out,8*i+j,val);
    }
  }
  return out;
}
static final int[] PC1 = {
  57, 49, 41, 33, 25, 17,  9,
   1, 58, 50, 42, 34, 26, 18,
  10,  2, 59, 51, 43, 35, 27,
  19, 11,  3, 60, 52, 44, 36,
  63, 55, 47, 39, 31, 23, 15,
   7, 62, 54, 46, 38, 30, 22,
  14,  6, 61, 53, 45, 37, 29,
  21, 13,  5, 28, 20, 12,  4
};
static final int[] PC2 = {
  14, 17, 11, 24,  1,  5,
   3, 28, 15,  6, 21, 10,
  23, 19, 12,  4, 26,  8,
  16,  7, 27, 20, 13,  2,
  41, 52, 31, 37, 47, 55,
  30, 40, 51, 45, 33, 48,
  44, 49, 39, 56, 34, 53,
  46, 42, 50, 36, 29, 32
};
static final int[] SHIFTS = {
  1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1
};
```

```
  private static byte[][] getSubkeys(byte[] theKey)
  throws Exception {
  int activeKeySize = PC1.length;
  int numOfSubKeys = SHIFTS.length;
  byte[] activeKey = selectBits(theKey,PC1);
  int halfKeySize = activeKeySize/2;
  byte[] c = selectBits(activeKey,0,halfKeySize);
  byte[] d = selectBits(activeKey,halfKeySize,halfKeySize);
  byte[][] subKeys = new byte[numOfSubKeys][];
  for (int k=0; k<numOfSubKeys; k++) {
    c = rotateLeft(c,halfKeySize,SHIFTS[k]);
    d = rotateLeft(d,halfKeySize,SHIFTS[k]);
    byte[] cd = concatenateBits(c,halfKeySize,d,halfKeySize);
    subKeys[k] = selectBits(cd,PC2);
  }
  return subKeys;
}
private static byte[] rotateLeft(byte[] in, int len, int step) {
  int numOfBytes = (len-1)/8 + 1;
  byte[] out = new byte[numOfBytes];
  for (int i=0; i<len; i++) {
    int val = getBit(in,(i+step)%len);
    setBit(out,i,val);
  }
  return out;
}
private static byte[] concatenateBits(byte[] a, int aLen, byte[] b,
  int bLen) {
  int numOfBytes = (aLen+bLen-1)/8 + 1;
  byte[] out = new byte[numOfBytes];
  int j = 0;
  for (int i=0; i<aLen; i++) {
    int val = getBit(a,i);
    setBit(out,j,val);
    j++;
  }
  for (int i=0; i<bLen; i++) {
    int val = getBit(b,i);
    setBit(out,j,val);
    j++;
  }
  return out;
}
private static byte[] selectBits(byte[] in, int pos, int len) {
  int numOfBytes = (len-1)/8 + 1;
  byte[] out = new byte[numOfBytes];
```

```
    for (int i=0; i<len; i++) {
      int val = getBit(in,pos+i);
      setBit(out,i,val);
    }
    return out;
  }
  private static byte[] selectBits(byte[] in, int[] map) {
    int numOfBytes = (map.length-1)/8 + 1;
    byte[] out = new byte[numOfBytes];
    for (int i=0; i<map.length; i++) {
      int val = getBit(in,map[i]-1);
      setBit(out,i,val);
    }
    return out;
  }
  private static int getBit(byte[] data, int pos) {
    int posByte = pos/8;
    int posBit = pos%8;
    byte valByte = data[posByte];
    int valInt = valByte>>(8-(posBit+1)) & 0x0001;
    return valInt;
  }
  private static void setBit(byte[] data, int pos, int val) {
    int posByte = pos/8;
    int posBit = pos%8;
    byte oldByte = data[posByte];
    oldByte = (byte) (((0xFF7F>>posBit) & oldByte) & 0x00FF);
    byte newByte = (byte) ((val<<(8-(posBit+1))) | oldByte);
    data[posByte] = newByte;
  }
  private static byte[] readBytes(String in) throws Exception {
    FileInputStream fis = new FileInputStream(in);
    int numOfBytes = fis.available();
    byte[] buffer = new byte[numOfBytes];
    fis.read(buffer);
    fis.close();
    return buffer;
  }
  private static void writeBytes(byte[] data, String out)
    throws Exception {
    FileOutputStream fos = new FileOutputStream(out);
    fos.write(data);
    fos.close();
  }
  private static void printBytes(byte[] data, String name) {
    System.out.println("");
```

```
    System.out.println(name+":");
    for (int i=0; i<data.length; i++) {
      System.out.print(byteToBits(data[i])+" ");
    }
    System.out.println();
  }
  private static String byteToBits(byte b) {
    StringBuffer buf = new StringBuffer();
    for (int i=0; i<8; i++)
      buf.append((int)(b>>(8-(i+1)) & 0x0001));
    return buf.toString();
  }
}
```

Decryption:

```
import java.io.*;
public class CipherDES {

        public static byte[] cryptDES(byte[] input,String key,String mode) throws Exception
        {
                byte[] theKey = key.getBytes();
                byte[][] subKeys = getSubkeys(theKey);

                ByteArrayInputStream bin=new ByteArrayInputStream(input);
                ByteArrayOutputStream bout=new ByteArrayOutputStream();
                byte[] c=new byte[8];

                while(  bin.read(c) !=-1 )
                {
                        //System.out.println("Before size "+c.length);
                        byte[] d=cipher(c,subKeys,mode);
                        bout.write(d);
                        c=null;
                        c=new byte[8];
                }
                return bout.toByteArray();
        }
    public static final int[] IP = {
     58, 50, 42, 34, 26, 18, 10, 2,
     60, 52, 44, 36, 28, 20, 12, 4,
     62, 54, 46, 38, 30, 22, 14, 6,
     64, 56, 48, 40, 32, 24, 16, 8,
     57, 49, 41, 33, 25, 17,  9, 1,
     59, 51, 43, 35, 27, 19, 11, 3,
```

```
    61, 53, 45, 37, 29, 21, 13, 5,
    63, 55, 47, 39, 31, 23, 15, 7
  };
 public static final int[] E = {
     32,  1,  2,  3,  4,  5,
      4,  5,  6,  7,  8,  9,
      8,  9, 10, 11, 12, 13,
     12, 13, 14, 15, 16, 17,
     16, 17, 18, 19, 20, 21,
     20, 21, 22, 23, 24, 25,
     24, 25, 26, 27, 28, 29,
     28, 29, 30, 31, 32,  1
  };
  public static final int[] P = {
     16,  7, 20, 21,
     29, 12, 28, 17,
      1, 15, 23, 26,
      5, 18, 31, 10,
      2,  8, 24, 14,
     32, 27,  3,  9,
     19, 13, 30,  6,
     22, 11,  4, 25
   };
 public   static final int[] INVP = {
     40, 8, 48, 16, 56, 24, 64, 32,
     39, 7, 47, 15, 55, 23, 63, 31,
     38, 6, 46, 14, 54, 22, 62, 30,
     37, 5, 45, 13, 53, 21, 61, 29,
     36, 4, 44, 12, 52, 20, 60, 28,
     35, 3, 43, 11, 51, 19, 59, 27,
     34, 2, 42, 10, 50, 18, 58, 26,
     33, 1, 41,  9, 49, 17, 57, 25
   };
   public static byte[] cipher(byte[] theMsg, byte[][] subKeys,
     String mode) throws Exception {
     if (theMsg.length<8)
       throw new Exception("Message is less than 64 bits.");
     theMsg = selectBits(theMsg,IP); // Initial Permutation
     int blockSize = IP.length;
     byte[] l = selectBits(theMsg,0,blockSize/2);
     byte[] r = selectBits(theMsg,blockSize/2,blockSize/2);
     int numOfSubKeys = subKeys.length;
     for (int k=0; k<numOfSubKeys; k++) {
         byte[] rBackup = r;
       r = selectBits(r,E); // Expansion
       if (mode.equalsIgnoreCase("encrypt"))
```

```
        r = doXORBytes(r,subKeys[k]); // XOR with the sub key
      else
        r = doXORBytes(r,subKeys[numOfSubKeys-k-1]);
      r = substitution6x4(r); // Substitution
      r = selectBits(r,P); // Permutation
      r = doXORBytes(l,r); // XOR with the previous left half
      l = rBackup; // Taking the previous right half
    }
    byte[] lr = concatenateBits(r,blockSize/2,l,blockSize/2);
    lr = selectBits(lr,INVP); // Inverse Permutation
    return lr;
  }

  public static byte[] doXORBytes(byte[] a, byte[] b) {
    byte[] out = new byte[a.length];
    for (int i=0; i<a.length; i++) {
      out[i] = (byte) (a[i] ^ b[i]);
    }
    return out;
  }
  public static final int[] S = {
14,  4, 13,  1,  2, 15, 11,  8,  3, 10,  6, 12,  5,  9,  0,  7, // S1
 0, 15,  7,  4, 14,  2, 13,  1, 10,  6, 12, 11,  9,  5,  3,  8,
 4,  1, 14,  8, 13,  6,  2, 11, 15, 12,  9,  7,  3, 10,  5,  0,
15, 12,  8,  2,  4,  9,  1,  7,  5, 11,  3, 14, 10,  0,  6, 13,
15,  1,  8, 14,  6, 11,  3,  4,  9,  7,  2, 13, 12,  0,  5, 10, // S2
 3, 13,  4,  7, 15,  2,  8, 14, 12,  0,  1, 10,  6,  9, 11,  5,
 0, 14,  7, 11, 10,  4, 13,  1,  5,  8, 12,  6,  9,  3,  2, 15,
13,  8, 10,  1,  3, 15,  4,  2, 11,  6,  7, 12,  0,  5, 14,  9,
10,  0,  9, 14,  6,  3, 15,  5,  1, 13, 12,  7, 11,  4,  2,  8, // S3
13,  7,  0,  9,  3,  4,  6, 10,  2,  8,  5, 14, 12, 11, 15,  1,
13,  6,  4,  9,  8, 15,  3,  0, 11,  1,  2, 12,  5, 10, 14,  7,
 1, 10, 13,  0,  6,  9,  8,  7,  4, 15, 14,  3, 11,  5,  2, 12,
 7, 13, 14,  3,  0,  6,  9, 10,  1,  2,  8,  5, 11, 12,  4, 15, // S4
13,  8, 11,  5,  6, 15,  0,  3,  4,  7,  2, 12,  1, 10, 14,  9,
10,  6,  9,  0, 12, 11,  7, 13, 15,  1,  3, 14,  5,  2,  8,  4,
 3, 15,  0,  6, 10,  1, 13,  8,  9,  4,  5, 11, 12,  7,  2, 14,
 2, 12,  4,  1,  7, 10, 11,  6,  8,  5,  3, 15, 13,  0, 14,  9, // S5
14, 11,  2, 12,  4,  7, 13,  1,  5,  0, 15, 10,  3,  9,  8,  6,
 4,  2,  1, 11, 10, 13,  7,  8, 15,  9, 12,  5,  6,  3,  0, 14,
11,  8, 12,  7,  1, 14,  2, 13,  6, 15,  0,  9, 10,  4,  5,  3,
12,  1, 10, 15,  9,  2,  6,  8,  0, 13,  3,  4, 14,  7,  5, 11, // S6
10, 15,  4,  2,  7, 12,  9,  5,  6,  1, 13, 14,  0, 11,  3,  8,
 9, 14, 15,  5,  2,  8, 12,  3,  7,  0,  4, 10,  1, 13, 11,  6,
 4,  3,  2, 12,  9,  5, 15, 10, 11, 14,  1,  7,  6,  0,  8, 13,
 4, 11,  2, 14, 15,  0,  8, 13,  3, 12,  9,  7,  5, 10,  6,  1, // S7
```

```
13, 0, 11, 7, 4, 9, 1, 10, 14, 3, 5, 12, 2, 15, 8, 6,
1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6, 8, 0, 5, 9, 2,
6, 11, 13, 8, 1, 4, 10, 7, 9, 5, 0, 15, 14, 2, 3, 12,
13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5, 0, 12, 7, // S8
1, 15, 13, 8, 10, 3, 7, 4, 12, 5, 6, 11, 0, 14, 9, 2,
7, 11, 4, 1, 9, 12, 14, 2, 0, 6, 10, 13, 15, 3, 5, 8,
2, 1, 14, 7, 4, 10, 8, 13, 15, 12, 9, 0, 3, 5, 6, 11
  };
  public static byte[] substitution6x4(byte[] in) {
    in = splitBytes(in,6); // Splitting byte[] into 6-bit blocks
    byte[] out = new byte[in.length/2];
    int lhByte = 0;
    for (int b=0; b<in.length; b++) { // Should be sub-blocks
      byte valByte = in[b];
      int r = 2*(valByte>>7&0x0001)+(valByte>>2&0x0001); // 1 and 6
      int c = valByte>>3&0x000F; // Middle 4 bits
      int hByte = S[64*b+16*r+c]; // 4 bits (half byte) output
      if (b%2==0) lhByte = hByte; // Left half byte
      else out[b/2] = (byte) (16*lhByte + hByte);
    }
    return out;
  }
  public static byte[] splitBytes(byte[] in, int len) {
    int numOfBytes = (8*in.length-1)/len + 1;
    byte[] out = new byte[numOfBytes];
    for (int i=0; i<numOfBytes; i++) {
      for (int j=0; j<len; j++) {
        int val = getBit(in, len*i+j);
        setBit(out,8*i+j,val);
      }
    }
    return out;
  }
  public static final int[] PC1 = {
    57, 49, 41, 33, 25, 17,  9,
     1, 58, 50, 42, 34, 26, 18,
    10,  2, 59, 51, 43, 35, 27,
    19, 11,  3, 60, 52, 44, 36,
    63, 55, 47, 39, 31, 23, 15,
     7, 62, 54, 46, 38, 30, 22,
    14,  6, 61, 53, 45, 37, 29,
    21, 13,  5, 28, 20, 12,  4
  };
  public  static final int[] PC2 = {
     14, 17, 11, 24,  1,  5,
      3, 28, 15,  6, 21, 10,
```

```
    23, 19, 12,  4, 26,  8,
    16,  7, 27, 20, 13,  2,
    41, 52, 31, 37, 47, 55,
    30, 40, 51, 45, 33, 48,
    44, 49, 39, 56, 34, 53,
    46, 42, 50, 36, 29, 32
  };
  static final int[] SHIFTS = {
    1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1
  };
  public static byte[][] getSubkeys(byte[] theKey)
      throws Exception {
      int activeKeySize = PC1.length;
      int numOfSubKeys = SHIFTS.length;
      byte[] activeKey = selectBits(theKey,PC1);
      int halfKeySize = activeKeySize/2;
      byte[] c = selectBits(activeKey,0,halfKeySize);
      byte[] d = selectBits(activeKey,halfKeySize,halfKeySize);
      byte[][] subKeys = new byte[numOfSubKeys][];
      for (int k=0; k<numOfSubKeys; k++) {
        c = rotateLeft(c,halfKeySize,SHIFTS[k]);
        d = rotateLeft(d,halfKeySize,SHIFTS[k]);
        byte[] cd = concatenateBits(c,halfKeySize,d,halfKeySize);
        subKeys[k] = selectBits(cd,PC2);
      }
      return subKeys;
    }
    public static byte[] rotateLeft(byte[] in, int len, int step) {
      int numOfBytes = (len-1)/8 + 1;
      byte[] out = new byte[numOfBytes];
      for (int i=0; i<len; i++) {
        int val = getBit(in,(i+step)%len);
        setBit(out,i,val);
      }
      return out;
    }
    public  static byte[] concatenateBits(byte[] a, int aLen, byte[] b,
      int bLen) {
      int numOfBytes = (aLen+bLen-1)/8 + 1;
      byte[] out = new byte[numOfBytes];
      int j = 0;
      for (int i=0; i<aLen; i++) {
        int val = getBit(a,i);
        setBit(out,j,val);
        j++;
      }
```

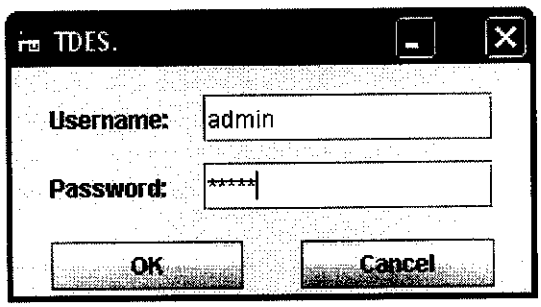
```
    for (int i=0; i<bLen; i++) {
      int val = getBit(b,i);
      setBit(out,j,val);
      j++;
    }
    return out;
  }
  public static byte[] selectBits(byte[] in, int pos, int len) {
    int numOfBytes = (len-1)/8 + 1;
    byte[] out = new byte[numOfBytes];
    for (int i=0; i<len; i++) {
      int val = getBit(in,pos+i);
      setBit(out,i,val);
    }
    return out;
  }
  public static byte[] selectBits(byte[] in, int[] map) {
    int numOfBytes = (map.length-1)/8 + 1;
    byte[] out = new byte[numOfBytes];
    for (int i=0; i<map.length; i++) {
      int val = getBit(in,map[i]-1);
      setBit(out,i,val);
    }
    return out;
  }
  public static int getBit(byte[] data, int pos) {
    int posByte = pos/8;
    int posBit = pos%8;
    byte valByte = data[posByte];
    int valInt = valByte>>(8-(posBit+1)) & 0x0001;
    return valInt;
  }
  public static void setBit(byte[] data, int pos, int val) {
    int posByte = pos/8;
    int posBit = pos%8;
    byte oldByte = data[posByte];
    oldByte = (byte) (((0xFF7F>>posBit) & oldByte) & 0x00FF);
    byte newByte = (byte) ((val<<(8-(posBit+1))) | oldByte);
    data[posByte] = newByte;
  }
  public static byte[] readBytes(String in) throws Exception {
    FileInputStream fis = new FileInputStream(in);
    int numOfBytes = fis.available();
    byte[] buffer = new byte[numOfBytes];
    fis.read(buffer);
    fis.close();
```

```
    return buffer;
  }
  public static void writeBytes(byte[] data, String out)
    throws Exception {
    FileOutputStream fos = new FileOutputStream(out);
    fos.write(data);
    fos.close();
  }
  public static void printBytes(byte[] data, String name) {
    System.out.println("");
    System.out.println(name+":");
    for (int i=0; i<data.length; i++) {
      System.out.print(byteToBits(data[i])+" ");
    }
    System.out.println();
  }
  public static String byteToBits(byte b) {
    StringBuffer buf = new StringBuffer();
    for (int i=0; i<8; i++)
      buf.append((int)(b>>(8-(i+1)) & 0x0001));
    return buf.toString();
  }
}
```

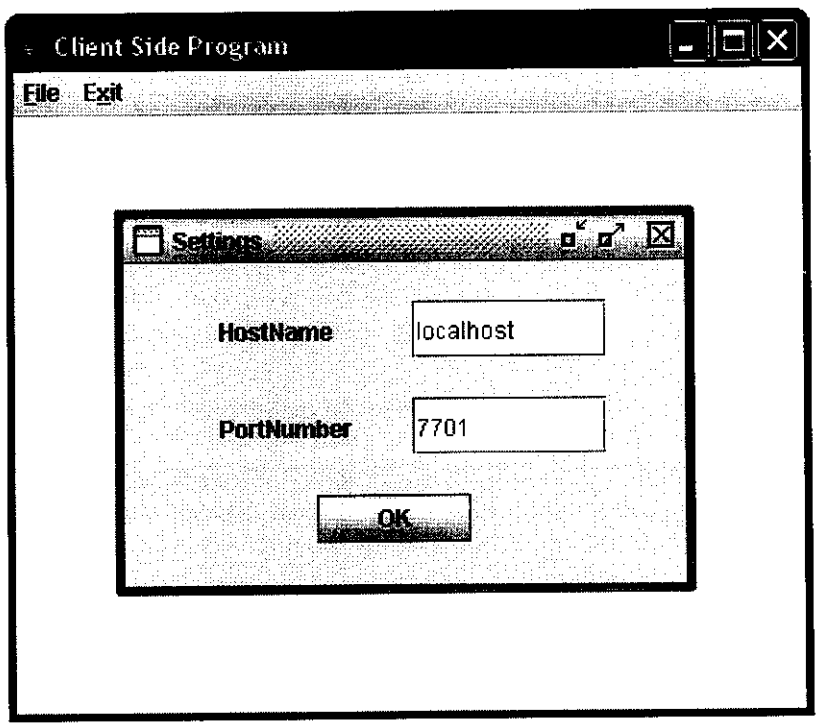
APPENDIX 2:

SCREEN SHOTS:

Admin Authentication:

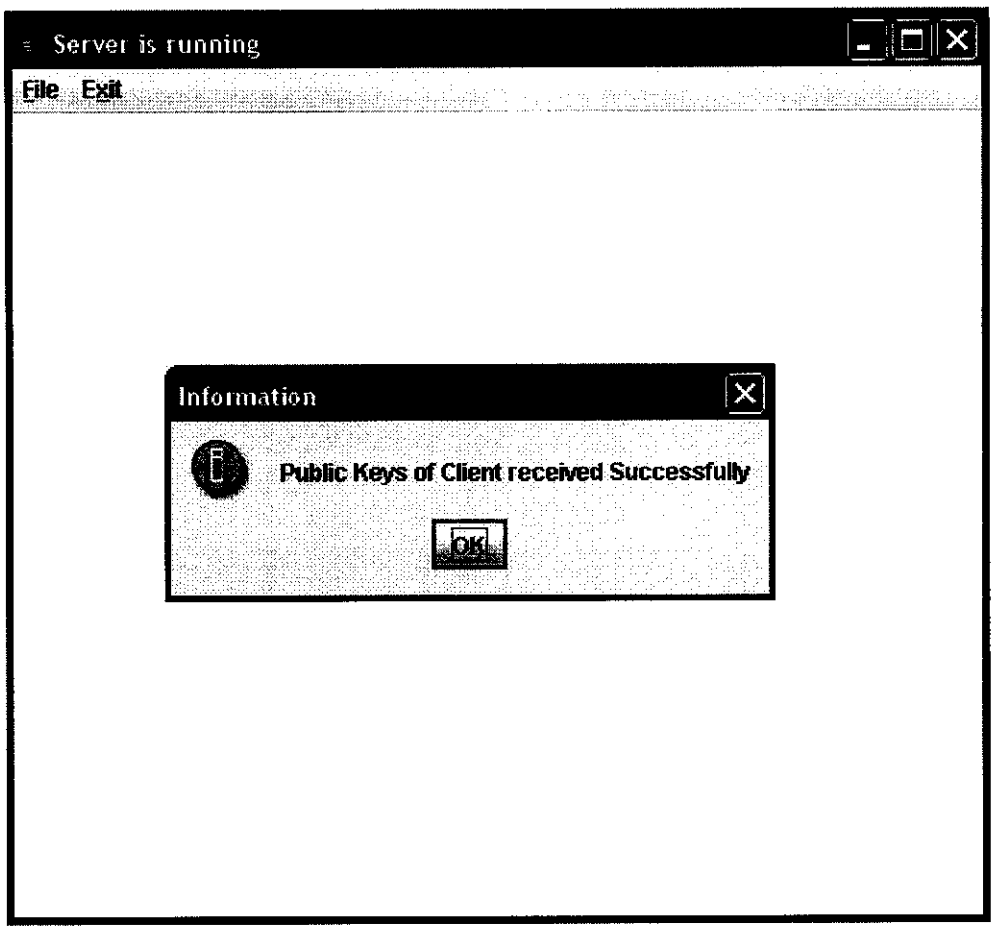


Connection Establishment:

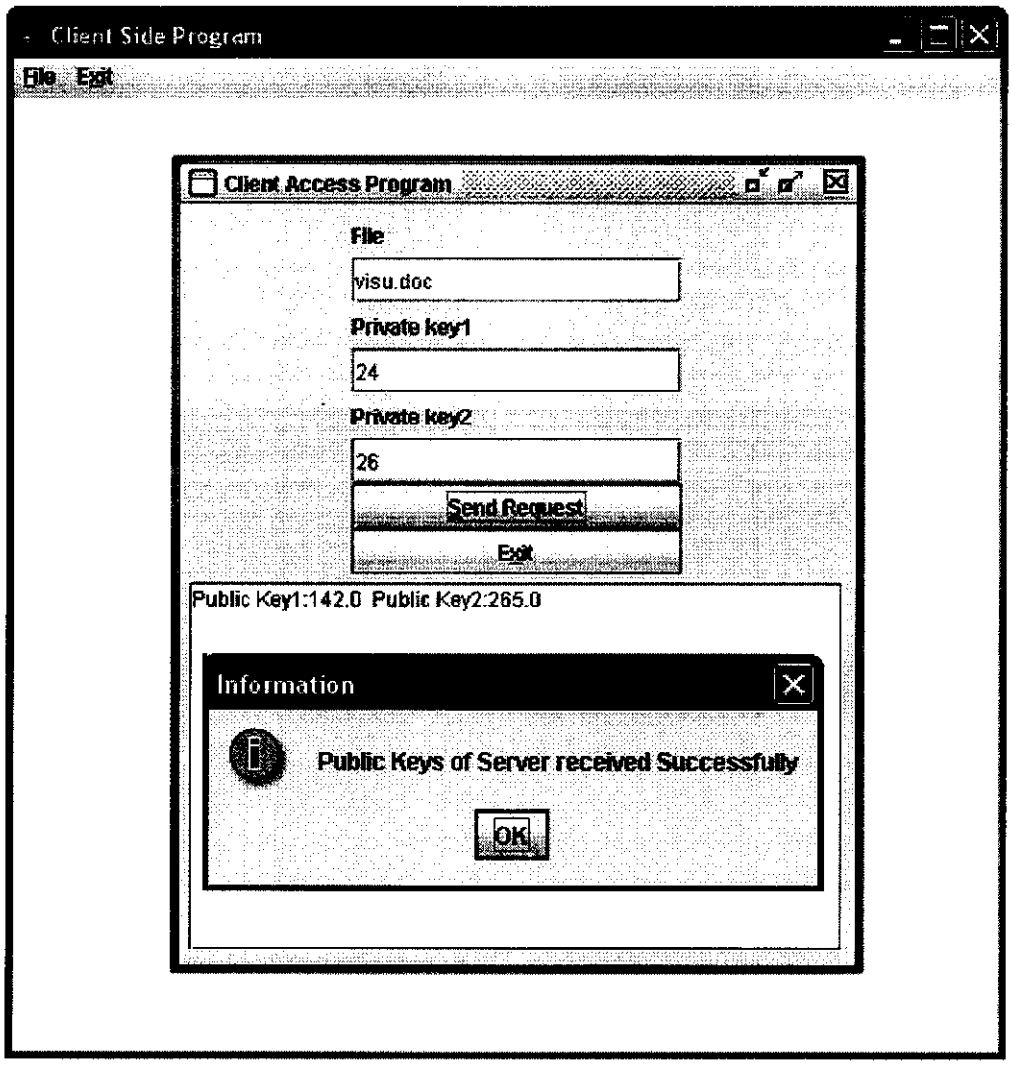


Diffie-Hellman KeyExchange:

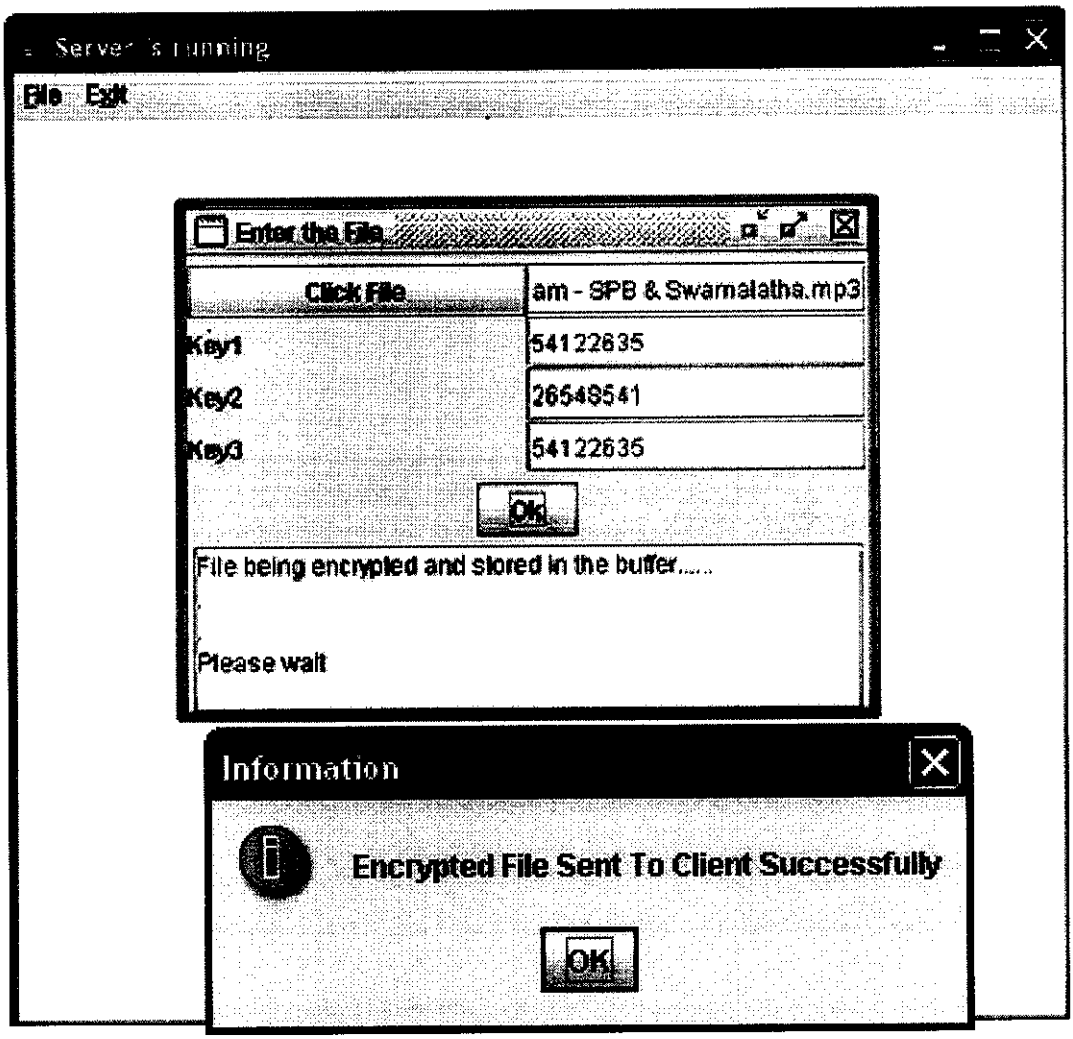
Server:



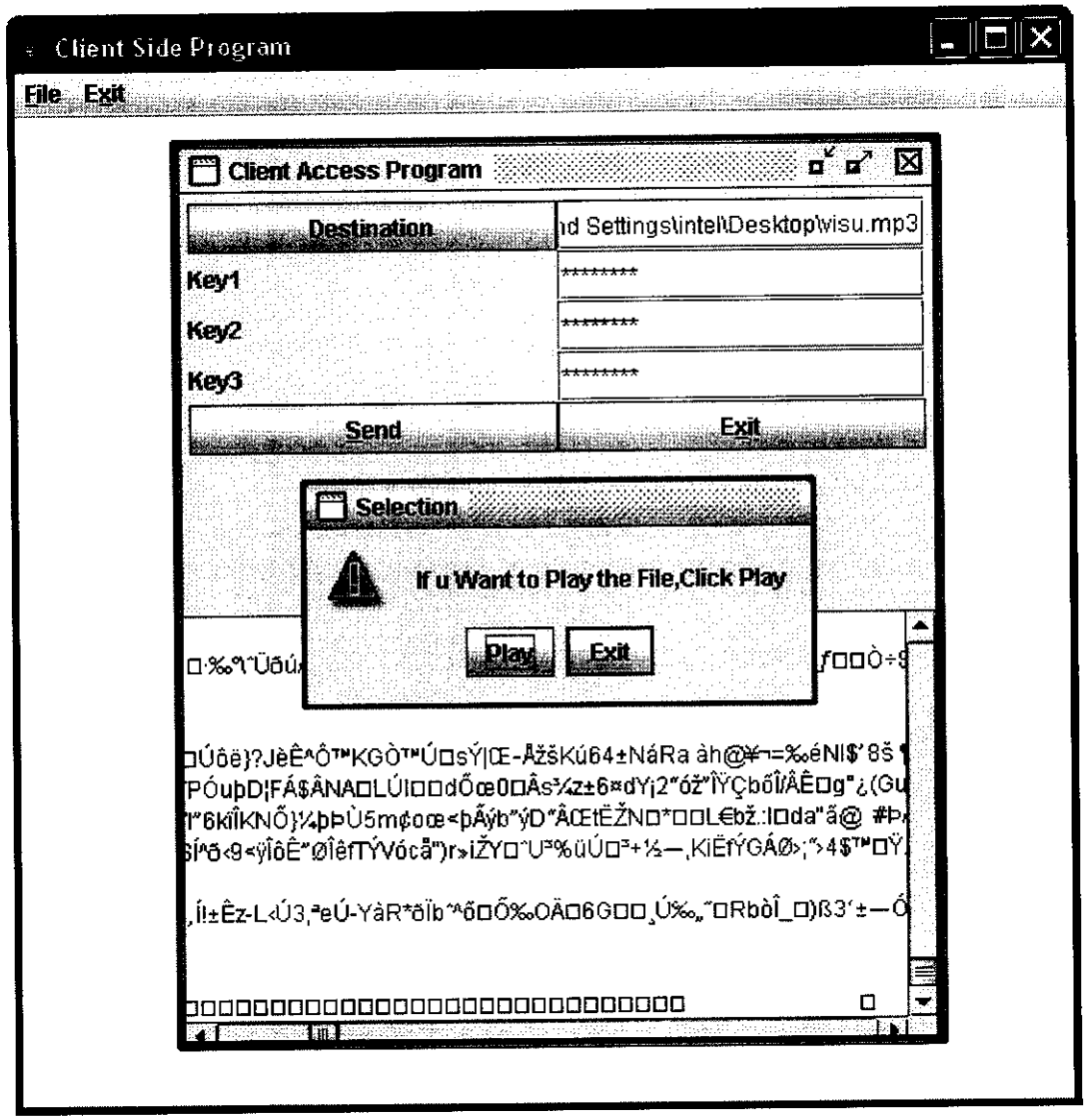
Client:


Client Side Program
File Exit
Client Access Program
File
visu.doc
Private key1
24
Private key2
26
Send Request
Exit
Public Key1:142.0 Public Key2:265.0
Information
Public Keys of Server received Successfully
OK

DES Server:


Server is running
File Exit
Enter the File
Click File
am - SPB & Swarnalatha.mp3
Key1
54122635
Key2
26548541
Key3
54122635
Ok
File being encrypted and stored in the buffer.....
Please wait
Information
Encrypted File Sent To Client Successfully
OK

DES Client:


Client Side Program
File Exit
Client Access Program
Destination
nd Settings\intel\Desktop\wisu.mp3
Key1
****
Key2
****
Key3
****
Send
Exit
Selection
If u Want to Play the File,Click Play
Play
Exit

APPENDIX 3:

DES EXAMPLE:

Let **M** be the plain text message **M** = 0123456789ABCDEF, where **M** is in hexadecimal (base 16) format. Rewriting **M** in binary format, we get the 64-bit block of text:

M = 0000 0001 0010 0011 0100 0101 0110 0111 1000 1001 1010 1011 1100 1101 1110 1111
L = 0000 0001 0010 0011 0100 0101 0110 0111
R = 1000 1001 1010 1011 1100 1101 1110 1111

The first bit of **M** is "0". The last bit is "1". We read from left to right.

DES operates on the 64-bit blocks using *key* sizes of 56- bits. The keys are actually stored as being 64 bits long, but every 8th bit in the key is not used (i.e. bits numbered 8, 16, 24, 32, 40, 48, 56, and 64). However, we will nevertheless number the bits from 1 to 64, going left to right, in the following calculations. But, as you will see, the eight bits just mentioned get eliminated when we create subkeys.

Example: Let **K** be the hexadecimal key **K** = 133457799BBCDFF1. This gives us as the binary key (setting 1 = 0001, 3 = 0011, etc., and grouping together every eight bits, of which the last one in each group will be unused):

K = 00010011 00110100 01010111 01111001 10011011 10111100 11011111 11110001

The DES algorithm uses the following steps:

Step 1: Create 16 subkeys, each of which is 48-bits long.

The 64-bit key is permuted according to the following table, **PC-1**. Since the first entry in the table is "57", this means that the 57th bit of the original key **K** becomes the first bit of the permuted key **K**+. The 49th bit of the original key becomes the second bit of the permuted key. The 4th bit of the original key is the last bit of the permuted key. Note only 56 bits of the original key appear in the permuted key.

PC-1

57	49	41	33	25	17	9
1	58	50	42	34	26	18
10	2	59	51	43	35	27
19	11	3	60	52	44	36
63	55	47	39	31	23	15
7	62	54	46	38	30	22
14	6	61	53	45	37	29
21	13	5	28	20	12	4

Example: From the original 64-bit key

K = 00010011 00110100 01010111 01111001 10011011 10111100 11011111 11110001

we get the 56-bit permutation

K+ = 1111000 0110011 0010101 0101111 0101010 1011001 1001111 0001111

Next, split this key into left and right halves, C_0 and D_0, where each half has 28 bits.

Example: From the permuted key **K+**, we get

C_0 = 1111000 0110011 0010101 0101111
D_0 = 0101010 1011001 1001111 0001111

With C_0 and D_0 defined, we now create sixteen blocks C_n and D_n, $1<=n<=16$. Each pair of blocks C_n and D_n is formed from the previous pair C_{n-1} and D_{n-1}, respectively, for n = 1, 2,..., 16, using the following schedule of "left shifts" of the previous block. To do a left shift, move each bit one place to the left, except for the first bit, which is cycled to the end of the block.

Iteration Number	Number of Left Shifts
1	1
2	1
3	2
4	2
5	2
6	2
7	2
8	2
9	1
10	2
11	2
12	2
13	2
14	2
15	2
16	1

This means, for example, C_3 and D_3 are obtained from C_2 and D_2, respectively, by two left shifts, and C_{16} and D_{16} are obtained from C_{15} and D_{15}, respectively, by one left shift. In all cases, by a single left shift is meant a rotation of the bits one place to the left, so that after one left shift the bits in the 28 positions are the bits that were previously in positions 2, 3,..., 28, 1.

Example: From original pair pair C_0 and D_0 we obtain:

C_0 = 1111000011001100101010101111
D_0 = 0101010101100110011110001111

C_1 = 1110000110011001010101011111
D_1 = 1010101011001100111100011110

C_2 = 1100001100110010101010111111
D_2 = 0101010110011001111000111101

C_3 = 0000110011001010101011111111
D_3 = 0101011001100111100011110101

C_4 = 0011001100101010101111111100
D_4 = 0101100110011110001111010101

C_5 = 1100110010101010111111110000
D_5 = 0110011001111000111101010101

C_6 = 0011001010101011111111000011
D_6 = 1001100111100011110101010101

C_7 = 1100101010101111111100001100
D_7 = 0110011110001111010101010110

C_8 = 0010101010111111110000110011
D_8 = 1001111000111101010101011001

C_9 = 0101010101111111100001100110
D_9 = 0011110001111010101010110011

C_{10} = 0101010111111110000110011001
D_{10} = 1111000111101010101011001100

C_{11} = 0101011111111000011001100101
D_{11} = 1100011110101010101100110011

C_{12} = 0101111111100001100110010101
D_{12} = 0001111010101010110011001111

C_{13} = 0111111110000110011001010101
D_{13} = 0111101010101011001100111100

C_{14} = 1111111000011001100101010101
D_{14} = 1110101010101100110011110001

C_{15} = 1111100001100110010101010111
D_{15} = 1010101010110011001111000111

C_{16} = 1111000011001100101010101111
D_{16} = 0101010101100110011110001111

We now form the keys K_n, for $1<=n<=16$, by applying the following permutation table to each of the concatenated pairs C_nD_n. Each pair has 56 bits, but **PC-2** only uses 48 of these.

PC-2

14	17	11	24	1	5
3	28	15	6	21	10
23	19	12	4	26	8
16	7	27	20	13	2
41	52	31	37	47	55
30	40	51	45	33	48
44	49	39	56	34	53
46	42	50	36	29	32

Therefore, the first bit of K_n is the 14th bit of C_nD_n, the second bit the 17th, and so on, ending with the 48th bit of K_n being the 32th bit of C_nD_n.

Example: For the first key we have C_1D_1 = 1110000 1100110 0101010 1011111 1010101 0110011 0011110 0011110

which, after we apply the permutation **PC-2**, becomes

K_1 = 000110 110000 001011 101111 111111 000111 000001 110010

For the other keys we have

K_2 = 011110 011010 111011 011001 110110 111100 100111 100101
K_3 = 010101 011111 110010 001010 010000 101100 111110 011001
K_4 = 011100 101010 110111 010110 110110 110011 010100 011101
K_5 = 011111 001110 110000 000111 111010 110101 001110 101000
K_6 = 011000 111010 010100 111110 010100 000111 101100 101111
K_7 = 111011 001000 010010 110111 111101 100001 100010 111100
K_8 = 111101 111000 101000 111010 110000 010011 101111 111011
K_9 = 111000 001101 101111 101011 111011 011110 011110 000001
K_{10} = 101100 011111 001101 000111 101110 100100 011001 001111
K_{11} = 001000 010101 111111 010011 110111 101101 001110 000110
K_{12} = 011101 010111 000111 110101 100101 000110 011111 101001
K_{13} = 100101 111100 010111 010001 111110 101011 101001 000001
K_{14} = 010111 110100 001110 110111 111100 101110 011100 111010
K_{15} = 101111 111001 000110 001101 001111 010011 111100 001010
K_{16} = 110010 110011 110110 001011 000011 100001 011111 110101

So much for the subkeys. Now we look at the message itself.

Step 2: Encode each 64-bit block of data.

There is an *initial permutation* **IP** of the 64 bits of the message data **M**. This rearranges the bits according to the following table, where the entries in the table show the new arrangement of the bits from their initial order. The 58th bit of **M** becomes the first bit of **IP**. The 50th bit of **M** becomes the second bit of **IP**. The 7th bit of **M** is the last bit of **IP**.

IP

58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

Example: Applying the initial permutation to the block of text **M**, given previously, we get

M = 0000 0001 0010 0011 0100 0101 0110 0111 1000 1001 1010 1011 1100 1101 1110 1111
IP = 1100 1100 0000 0000 1100 1100 1111 1111 1111 0000 1010 1010 1111 0000 1010 1010

Here the 58th bit of **M** is "1", which becomes the first bit of **IP**. The 50th bit of **M** is "1", which becomes the second bit of **IP**. The 7th bit of **M** is "0", which becomes the last bit of **IP**.

Next divide the permuted block **IP** into a left half L_0 of 32 bits, and a right half R_0 of 32 bits.

Example: From **IP**, we get L_0 and R_0

L_0 = 1100 1100 0000 0000 1100 1100 1111 1111
R_0 = 1111 0000 1010 1010 1111 0000 1010 1010

We now proceed through 16 iterations, for $1<=n<=16$, using a function f which operates on two blocks--a data block of 32 bits and a key K_n of 48 bits--to produce a block of 32 bits. **Let + denote XOR addition, (bit-by-bit addition modulo 2)**. Then for **n** going from 1 to 16 we calculate

$$L_n = R_{n-1}$$
$$R_n = L_{n-1} + f(R_{n-1}, K_n)$$

This results in a final block, for $n = 16$, of $L_{16}R_{16}$. That is, in each iteration, we take the right 32 bits of the previous result and make them the left 32 bits of the current step. For the right 32 bits in the current step, we XOR the left 32 bits of the previous step with the calculation f.

Example: For $n = 1$, we have

K_1 = 000110 110000 001011 101111 111111 000111 000001 110010
$L_1 = R_0$ = 1111 0000 1010 1010 1111 0000 1010 1010
$R_1 = L_0 + f(R_0, K_1)$

It remains to explain how the function f works. To calculate f, we first expand each block R_{n-1} from 32 bits to 48 bits. This is done by using a selection table that repeats some of the bits in R_{n-1}. We'll call the use of this selection table the function **E**. Thus $\mathbf{E}(R_{n-1})$ has a 32 bit input block, and a 48 bit output block.

Let **E** be such that the 48 bits of its output, written as 8 blocks of 6 bits each, are obtained by selecting the bits in its inputs in order according to the following table:

E BIT-SELECTION TABLE

32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

Thus the first three bits of $\mathbf{E}(R_{n-1})$ are the bits in positions 32, 1 and 2 of R_{n-1} while the last 2 bits of $\mathbf{E}(R_{n-1})$ are the bits in positions 32 and 1.

Example: We calculate $\mathbf{E}(R_0)$ from R_0 as follows:

R_0 = 1111 0000 1010 1010 1111 0000 1010 1010
$\mathbf{E}(R_0)$ = 011110 100001 010101 010101 011110 100001 010101 010101

(Note that each block of 4 original bits has been expanded to a block of 6 output bits.)

Next in the f calculation, we XOR the output $\mathbf{E}(R_{n-1})$ with the key K_n:

$$K_n + \mathbf{E}(R_{n-1}).$$

Example: For K_1, $\mathbf{E}(R_0)$, we have

K_1 = 000110 110000 001011 101111 111111 000111 000001 110010
$\mathbf{E}(R_0)$ = 011110 100001 010101 010101 011110 100001 010101 010101
$K_1 + \mathbf{E}(R_0)$ = 011000 010001 011110 111010 100001 100110 010100 100111.

We have not yet finished calculating the function f. To this point we have expanded R_{n-1} from 32 bits to 48 bits, using the selection table, and XORed the result with the key K_n. We now have 48 bits, or eight groups of six bits. We now do something strange with each group of six bits: we use them as addresses in tables called "**S boxes**". Each group of six bits will give us an address in a different S box. Located at that address will be a 4 bit number. This 4

bit number will replace the original 6 bits. The net result is that the eight groups of 6 bits are transformed into eight groups of 4 bits (the 4-bit outputs from the S boxes) for 32 bits total.

Write the previous result, which is 48 bits, in the form:

$$K_n + E(R_{n-1}) = B_1B_2B_3B_4B_5B_6B_7B_8,$$

where each B_i is a group of six bits. We now calculate

$$S_1(B_1)S_2(B_2)S_3(B_3)S_4(B_4)S_5(B_5)S_6(B_6)S_7(B_7)S_8(B_8)$$

where $S_i(B_i)$ referres to the output of the i-th S box.

To repeat, each of the functions $S1, S2,..., S8$, takes a 6-bit block as input and yields a 4-bit block as output. The table to determine S_1 is shown and explained below:

S1

Column Number

Row No.	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
2	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

If S_1 is the function defined in this table and B is a block of 6 bits, then $S_1(B)$ is determined as follows: The first and last bits of B represent in base 2 a number in the decimal range 0 to 3 (or binary 00 to 11). Let that number be i. The middle 4 bits of B represent in base 2 a number in the decimal range 0 to 15 (binary 0000 to 1111). Let that number be j. Look up in the table the number in the i-th row and j-th column. It is a number in the range 0 to 15 and is uniquely represented by a 4 bit block. That block is the output $S_1(B)$ of S_1 for the input B. For example, for input block B = 011011 the first bit is "0" and the last bit "1" giving 01 as the row. This is row 1. The middle four bits are "1101". This is the binary equivalent of decimal 13, so the column is column number 13. In row 1, column 13 appears 5. This determines the output; 5 is binary 0101, so that the output is 0101. Hence $S_1(011011) = 0101$.

The tables defining the functions $S_1,...,S_8$ are the following:

S1

14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

S2

15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10

3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9

S3

10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12

S4

7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14

S5

2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3

S6

12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13

S7

4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12

S8

13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

Example: For the first round, we obtain as the output of the eight S boxes:

$K_1 + \mathbf{E}(R_0) = 011000\ 010001\ 011110\ 111010\ 100001\ 100110\ 010100\ 100111.$

$S_1(B_1)S_2(B_2)S_3(B_3)S_4(B_4)S_5(B_5)S_6(B_6)S_7(B_7)S_8(B_8) = 0101\ 1100\ 1000\ 0010\ 1011\ 0101\ 1001\ 0111$

The final stage in the calculation of f is to do a permutation **P** of the S-box output to obtain the final value of f:

$$f = \mathbf{P}(S_1(B_1)S_2(B_2)...S_8(B_8))$$

The permutation **P** is defined in the following table. **P** yields a 32-bit output from a 32-bit input by permuting the bits of the input block.

P

16	7	20	21
29	12	28	17
1	15	23	26
5	18	31	10
2	8	24	14
32	27	3	9
19	13	30	6
22	11	4	25

Example: From the output of the eight **S** boxes:

$S_1(B_1)S_2(B_2)S_3(B_3)S_4(B_4)S_5(B_5)S_6(B_6)S_7(B_7)S_8(B_8)$ = 0101 1100 1000 0010 1011 0101 1001 0111

we get

$$f = 0010\ 0011\ 0100\ 1010\ 1010\ 1001\ 1011\ 1011$$

$R_1 = L_0 + f(R_0, K_1)$

= 1100 1100 0000 0000 1100 1100 1111 1111
+ 0010 0011 0100 1010 1010 1001 1011 1011
= 1110 1111 0100 1010 0110 0101 0100 0100

In the next round, we will have $L_2 = R_1$, which is the block we just calculated, and then we must calculate $R_2 = L_1 + f(R_1, K_2)$, and so on for 16 rounds. At the end of the sixteenth round we have the blocks L_{16} and R_{16}. We then *reverse* the order of the two blocks into the 64-bit block

$$R_{16}L_{16}$$

and apply a final permutation $\mathbf{IP}^{-1}$ as defined by the following table:

IP^{-1}

40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

That is, the output of the algorithm has bit 40 of the preoutput block as its first bit, bit 8 as its second bit, and so on, until bit 25 of the preoutput block is the last bit of the output.

Example: If we process all 16 blocks using the method defined previously, we get, on the 16th round,

L_{16} = 0100 0011 0100 0010 0011 0010 0011 0100
R_{16} = 0000 1010 0100 1100 1101 1001 1001 0101

We reverse the order of these two blocks and apply the final permutation to

$R_{16}L_{16}$ = 00001010 01001100 11011001 10010101 01000011 01000010 00110010 00110100

IP^{-1} = 10000101 11101000 00010011 01010100 00001111 00001010 10110100 00000101

which in hexadecimal format is

85E813540F0AB405.

This is the encrypted form of **M** = 0123456789ABCDEF: namely, **C** = 85E813540F0AB405.

Decryption is simply the inverse of encryption, follwing the same steps as above, but reversing the order in which the subkeys are applied.

REFERENCES

7. REFERENCES:

- Alfred J. Menezes, Paul C. Van Oorschot, and Scott A. Vanstone. (1997) '*Handbook of Applied Cryptography*', CRC Press, Boca Raton.
- Bruce Schneier. (1996) '*Applied Cryptography*', *Second Edition*, John Wiley & Sons, New York.
- Carl H. Meyer and Stephen M. Matyas.(1982) '*Cryptography: A New Dimension in Computer Data Security*', John Wiley & Sons, New York.
- Dorthy Elizabeth Robling Denning. .(1982), *Cryptography and Data Security*, Addison-Wesley Publishing Company, Reading, Massachusetts.
- Douglas R. Stinson. (1995) '*Cryptography: Theory and Practice*', CRC Press, Boca Raton.
- D.W. Davies and W.L. Price.(1989) '*Security for Computer Networks: An Introduction to Data Security in Teleprocessing and Electronics Funds Transfer*', Second Edition, John Wiley & Sons, New York.
- Miles E. Smid and Dennis K. Branstad. (1992) 'The Data Encryption Standard: Past and Future', in Gustavus J. Simmons, ed., *Contemporary Cryptography: The Science of Information Integrity*, IEEE Press.
- Orlin Grabbe. (2000) 'Illustration of DES Algorithm', Laissez Faire City Times, Vol 2, No. 28.
- E.Biham, L.R.Knudsen, 'DES, Triple-DES and AES', RSA CryptoBytes - Volume4, Number1, Summer 1998.
- D.Boneh (1998) 'Twenty years of Attacks on the RSA Cryptosystem', Stanford University,.
- "Certicom responds to recent attack on RSA Security System", Certicom, 6th May 1999.
- DATA ENCRYPTION STANDARD (DES)", NIST Federal Information Processing Standards Publication 46-3, Draft, 1999.
- M.J.B.Robshaw, "Security Estimates for 512-bit RSA", RSA Labs, 29th June 1995.
- Stinson, D. (2002) 'Cryptography: Theory and Practice'. Boca Raton, FL:CRC Press.
- http://www.webopedia.com
- http://www.java.sun.com