# File Security based on Pretty Good Privacy (PGP) Concept

Kamarudin Shafinah (Corresponding author) & Mohammad Mohd Ikram Faculty of Agriculture and Food Sciences, Universiti Putra Malaysia Bintulu Sarawak Campus (UPMKB) Nyabau Road, P.O. Box 396, 97008 Bintulu, Sarawak, Malaysia Tel: 60-086-855-328 E-mail: fienah2000@yahoo.com; ikram\_upmkb@yahoo.co.uk

Received: March 29, 2011 Accepted: June 21, 2011 doi:10.5539/cis.v4n4p10

## Abstract

Most computer systems are currently being integrated into networks. The purpose of integrating a computer system into a network is to make it more effective and easy to access. Various data and information are usually stored within a file in a server of this integrated computer network system. On the other hand, the data, information and file stored in a server might have the potential of being violated by intruders. The concept of Pretty Good Privacy (PGP) can relatively deal with the issues of computer security and to date, is still being used for applications such as electronic mails. Thus, in this study, the PGP concept was applied to enhance the security level for a digital file associated with a common text. This paper views the concept of PGP in general. The methodology used for applying the PGP concept as the security program for a digital file was explained. The changes on the digital file content for each changing processes of cryptography and compression were presented as the results in this paper.

Keywords: File security, Pretty good privacy, Cryptography, Encryption, Decryption and compression

## 1. Introduction

Countless organizations worldwide are continuously changing their methods of filing from paper based to computerized based systems for the storing of important data and information. To our knowledge, a file can generally be defined as a place to store various data and information. In certain cases, a file may contain confidential data and information like matters linking to banking, military and business sectors. A great deal of this scenario would therefore require a file of such importance to have a higher level of security in order to prevent its content from being accessed by unwanted intruders. Moreover, since most organizations nowadays have begun implementing and integrating their computer systems into a network, people subsequently have become evermore concern and aware on issues regarding file security and privacy. Further, by using a network, the concept of client/server can be achieved to a broader extent where users act as clients and all the data, information and file is stored in a server. Upon the clients request to obtain any data, information or file from a server, the server will immediately trace the requested data, information or file to later be sent back to the client as a manner of fulfilling the request of the client. While the network seems beneficial to the client, it does imminently increase the potential risk of an intruder contravening file security and privacy.

In a developing country such as Malaysia, a majority of its local organizations remain using firewall alone as a method of protecting their server from the intrusion issues. However, it can be delineated that being dependent via this method alone for enhancing the security level without having other additional protection might not be adequate enough to protect the server from being accessed by intruders. Once an intruder breaches through the firewall, they can easily have access to the content stored in the server. These so called intruders can hold strong initiatives of finding all sorts of methods, techniques and approaches to access the data, information or files that they intend to acquire. For instance, while delivering a digital file where the content of the file uses a common and readable text, an intruder may seize the opportunity to make interruption and modification to its initial content. The intruder will then send back the file with an altered content as a fake file to the target receiver. This consequently can cause problems after the receiver receives the fake file attachment believing that they are receiving an original attachment.

The concept of Pretty Good Privacy (PGP) is associated with issues of computer security. While there have been numerous developments to ensure computer security such as Intrusion Prevention, network device hardening, router based firewalls and etc., the PGP concept may also partially aid in providing valuable improvements to issues concerning computer security. This paper aims to explain the applying of PGP concept to enhance the

security level of a digital file with a common text. The paper included the methodology used in implementing the security program to secure a digital file. The paper can be used as a baseline reference for future implementation and application as well as in supporting to an increasing international reference database on matters relating to computer security.

## 2. General Review

Cryptography is related to computer security. The word cryptography was originally derived from the Greek words of *kryptos* and *graphos*. *Kryptos* is defined as secret whereas *graphos* is defined as writing (Bacard, 1995). Cryptography involves two processes which are encryption (scramble) and decryption (unscramble). Cryptography is a process of converting plaintext into ciphertext, and in contrast ciphertext into plaintext (Haney, 2006). Plaintext is a term that refers to an original text. There are certain mathematical formulae or rules that can be used for the encryption and decryption processes. These mathematical formulae or rules are known as cipher.

The following is an example of a cryptography concept:

Cipher: c=x, e=v, r=i, s=h, t=g

The example shows that the cipher for the letter 'c' is assigned with the letter 'x'. This denotes that whenever a user enters a letter 'c' as an input to a security program, this letter 'c' will automatically be converted to the letter 'x'. As another example, a user may want to enter the word 'secret' as the plaintext. After the user proceeds with the encryption process, this word will be transformed to 'hvxivg' as referring to the cipher examples given. This unreadable form of text is known as ciphertext. Users can also carry out decryption process to convert the ciphertext, 'hvxivg' back to plaintext, 'secret'.

Pretty Good Privacy (PGP) involves the process of cryptography. The PGP has commonly been used for electronic mails. The PGP concept was created by Phil Zimmermann in 1991 but did not have the unique techniques for encryption (Henry, 2000). Nonetheless, RSA, IDEA and digital signatures are the frequently used techniques for the encryption process in PGP at present. The PGP is widely used due to several advantages it pertain which among them being a freeware, existence on web, and more secure algorithms. The PGP can also be implemented into an assortment of platforms including DOS/Windows, UNIX and Macintosh. In addition, PGP is independent in view of the fact that it is neither in extensive development, nor is it controlled by any government or organizational standards.

Stallings (2002) explained on five operations involved in PGP: (1) authentication, (2) confidentiality, (3) compression, (4) email compatibility and (5) segmentation. Table 1 shows a summarize explanation of the aforementioned operations. Simultaneously, Triple Data Encryption Standard (TDES) algorithm was identified as one of the algorithms used in the confidentiality operation (Table 1). The TDES is the most common algorithm to be used for PGP. Triple Data Encryption Standard (TDES) was pioneered by Tuchman and the first encryption standard used in a financial application in ANSI X9.17 in 1985. According to the literature, TDES was later acknowledged as an encryption standard in 1999 with the announcement of FIPS PUB 46-3 (Data Encryption Standard, 1999; Stallings, 2002). The TDES algorithm was chosen for use in the development of this security program to secure a digital file. Triple Data Encryption Standard (TDES) uses three keys and three phases for the DES algorithms. Following is the function of encrypt-encrypt (EDE):

 $C = E_{K3}[D_{K2}[E_{K1}[P]]]$ 

where,

C = ciphertextP = plaintext

 $E_K[X] = encryption \text{ of } X \text{ using } K \text{ key}$ 

 $D_{K}[Y] = decryption \text{ of } Y \text{ using } K \text{ key}$ 

Decryption is a reverse process:

 $P = D_{K1}[E_{K2}[D_{K3}[C]]]$ 

The process of encryption and decryption for TDES is illustrated in Fig. 1. There is no cryptographic benefit in using decryption as second phase (Fidanci et al., 2002). The advantage of TDES is its capability of decrypting data using the pervious DES because the previous DES utilizes 56 bit keys for the cryptography process (Data Encryption Standard, 1999; Stallings, 2002).

 $C = E_{K1}[D_{K1}[E_{K1}[P]]] = E_{K1}[P]$ 

TDES has 168 bit keys and FIPS 46-3 gives permission for two keys where K1=K2 with 112 bit keys. On the whole, this general review on cryptography, PGP and TDES provides ample brief information in facilitating and applying the PGP concept to enhance the security level for a digital file.

## 3. Methodology

Four phases were involved in applying the PGP concept as the security program for a digital file. The phases comprised of: (1) preliminary investigation, (2) analysis and design, (3) implementation and (4) testing.

## 3.1 Preliminary Investigation

This study was conducted in 2003. At the time, many Malaysian organizations and individuals were less aware and concern about the security level of digital data, information or file except for the banking institution. Much of this changed when local organizations started integrating their systems into computer networking environments. This led to the issues of insecure computer security in particular for confidential digital data, information and file. Hence, it is with this insight that the study aimed in implementing a security program to enhance the security level of a digital file. Seeing that a file can be a vital place to store data and information, a digital file was put into consideration and attention. Then again, the development of this security program was restricted for use only for a digital file with a common text. The concept of PGP was selected for application as an additional measure to increase the security level of a digital file. Through this additional measure, intruders will have to endure more difficulties when trying to access the files content.

Apart from that, information on the requirements for implementing a security program throughout the preliminary investigation was obtained from books, journals and internet sources. However, it should be pointed out that the attaining of substantial information that can be used as references especially from books was the most intricate task due to the limited and scarce number of computer security books that were commercially available in Malaysia. The limitation is common in a number of developing countries owing to the restriction in the selling and sharing of computer security books to the global society which instead was rather confined within certain country boundaries.

# 3.2 Analysis and Design

For this phase, an analysis on the flow sequence for the security program was carried out. The core design and development of this security program was based on the PGP concept which is illustrated in Fig. 2. The sender part entails encryption and compression (Zip) in a way to convert the content of a digital file from plaintext into ciphertext. The ciphertext is then compressed. In the meantime, the receiver part carries out the decompression (Unzip) and decryption as a method to access the original content (plaintext) of the encrypted digital file. The security program uses a similar key for both encryption and decryption processes. On both part, the sender and receiver were required to install the same key within their computer prior to using the security program. The use of a similar key for both encryption is known as the conventional PGP method.

The process of delivering digital file from sender to receiver was also carried out. Therefore, the security program was required to use Transmission Control Protocol/Internet Protocol (TCP/IP). At this point, the concept of client/server was applied to accomplish the process of file delivering. Generally, the host of TCP/IP can be specific to 65535 ports and the standard port for File Transfer Protocol (FTP) is port 21. On the other hand, the implementation of this security program was not specified to any port number. Both server and client must thus decide their port number before they can establish a network. As shown in Fig. 3, both client and server can play a role as either the sender or the receiver. For example, if a server sends a message, it will designate that the server acted the role as a sender and the client will become the receiver.

The entire architecture of the security program including both sender and receiver part is depicted in Fig. 4. Based on Fig. 4, the sender needs to browse and choose at a file that was intended to be sent. If the file does not require a high level of security, then the sender can proceed with the sending process directly. If in circumstances the file requires a high level of security, then the sender should proceed in selecting the Encrypt button. Thereof, the process of encryption and compression is automatically generated. Through the encryption and compression processes, the file with the extension of .enc and .zip will be created. Sender should select the file with the extension of .zip because it contains the complete enhancement of the security program to be submitted to the receiver. On the receiver part, a notification will appear whereby the receiver needs to decide whether to accept or reject the file sent. If the receiver agrees not to decline the file, the receiver will receive the sent file. The receiver should then explore the content of the file and identify whether the file needs to be decompressed and decrypted in a way to read the file content.

# 3.3 Implementation

For this phase, a suitable hardware and software was identified and selected. Java language was used to code the security programs involved. The security programs were: (1) a program for creating the secret key (Appendix A) and (2) a program which consists of cryptography and compression processes (Appendix B). The security program was coded to perform cryptography and compression processes mainly for a file with plaintext content. Java language was selected as the programming language because it has the capability to run under few platforms such as UNIX and Windows. The Java programming language is also capable of establishing connection amongst client and server (Darby et al., 2001; Deitel and Deitel, 2001; Deitel, 2002; Bronson, 2002). Compilation of these security programs was done using compiler j2sdk1.4.0. The hardware requirements needed for the implementation phase was a minimum of 4MB of memory and CPU Pentium 4 with 1.60GHz speed.

## 3.4 Testing

For this phase, the use of one computer was sufficient when testing the security program. The single computer executed the role of being both the client and server since the security program required an establishment of a connection between client and server. The testing phase comprised three which were: (1) cryptography and compression codes testing, (2) file delivering testing and (3) the input testing. Testing of the cryptography and compression codes were made to ensure that it produced an anticipated output which was the text inside the file being capable of encryption, compression, decryption and decompression. File delivering testing was conducted to ensure that the process of delivering a file from sender to receiver could be successfully accomplished. Lastly, the input testing was conducted to identify that the security program is capable of providing an error notification once a user enters an inappropriate input for the program.

## 4. Results

The results section describes the results from the testing phase. Before using the security program, the user needs to have the secret key. CryptKey.java is a program that was created for generating a secret key (Appendix A). Firstly, the Java version jdk1.4.0 was run on the computer. The command 'java CryptKey <key value>' was then inserted into the interface. The key value of 168 or 112 should be given as the input from the user to the CryptKey.Java program because the security program uses TDES algorithms. Fig. 5 shows an example of the secret key which was generated by using the key value of 112. The secret key was later stored on both server and client.

PGPftp.java is the security program which consists of the part for establishing a connection (client/server), cryptography, compression and file delivering (Appendix B). To run the program, 'Java PGPftp' command was inserted into the interface. The PGPftp.java program proceeded then with the establishment of a connection between server and client. Based on Fig. 6, two operations appeared on the screen where option 1 was referred to as client and option 2 was referred to as server. The user needs to decide whether wanting to act as the server or the client. Fig. 6 shows that option 2 was selected and the port number was required to be entered by the server. Meanwhile, Fig. 7 shows the client part. The client was required to enter the host address and port number (Fig. 7). Fig. 7 shows that the host address, 127.0.0.1 was given for the testing due to the security program which was run under the same computer. Furthermore, the port number that was entered by the client and server should be identical or similar to ensure that connections are successful. After a connection had been established between client and server, the interfaces for client and server appears on the screen (Fig. 8). The appearance of these interfaces showed that the testing to establish connection between the server and client or sender and receiver was successful.

The security program was tested next for encryption, compression, decompression, decryption and file delivering processes. The user needs to browse a file that was intentionally aimed at increasing files security level by using a Browse button indicated on the interface. After browsing and selecting a file, the user needs to click on the Encrypt button to proceed with the encryption and compression process. The program will request the user to enter a new file name with an extension <.enc> and this new file is used to store encrypted text content. The user then needs to enter a new file name with the extension <.zip> where the file was used to store the file content which had been compressed. The process of converting the encrypted file into a compressed file is automatically generated by PGPftp.java program. A notification appears on the sender part whenever the process was completed. The sender is then required to browse again and select the compressed file for the delivering process. The Send button is or should be clicked to proceed with the delivering was successful. On the receiver part, an authentication message will appear and the receiver is asked to accept or reject the file. The authentication message constituted of the senders name and file name. As soon as the receiver agrees to accept a file, a

notification message of recipient or file received will appear.

To read the content of the digital file that had been enhanced with the security level, the receiver needs to browse the file, followed by a click of a Decrypt button. The security program will then process the request of the receiver and ask the user to enter a new file name for the decompressed file in addition to a new file name for the decrypted file (Fig. 9). A notification message will appear to notify the receiver that the decompression and decryption processes were successful. Finally, the receiver can be able to read the content of the file. Fig. 10 shows the changes in the file content after each completion of the different processes such as encryption, compression, decompression and decryption. These changes were an indication of a functioning security program.

#### 5. Conclusion

Pretty Good Privacy (PGP) concept can be applied to increase the level of security for a digital file. With the security enhancement, intruders will have to deal with more obstacles in order to obtain the files content. The security program does not set up an exact port number between client and server due to the reason that this action can also be an additional obstruction for intruders when trying to gain access with the connection. The intruders must have the urge of guessing the correct port number before an interruption and modification can be made. Further, the digital file becomes more secure with the applying of cryptography and compression processes. The combination of cryptography and compression for the security program was mainly based on the PGP concept. The intruders would have to necessitate more time period, additional methods and a specific access key because of the difficulties when trying to access the files content as result of cryptography and compression. Besides, the method of delivering a file with the compression format <.zip> as compared to using the encrypted file with the extension <.enc> could be one way to masquerade the viewing of confidential files by intruders. The intruders may assume that the compression file was a normal file which did not contain any confidential or privacy data or information. Overall, the PGP concept can be pooled with other security methods or enhanced henceforward by network security officers and programmers as to potentially improve the computer security level of any respective organizations.

#### Acknowledgements

The first author wishes to acknowledge sincere thanks to Mohd Noor Derahman (Faculty of Computer Science and Information Technology, Universiti Putra Malaysia) and Dr. Osumanu Haruna Ahmed (Faculty Agriculture and Food Sciences, Universiti Putra Malaysia Bintulu Campus) rendering their technical support and assistance. The authors also thank the anonymous reviewers for their helpful comments.

### References

Bacard, A. (1995). The computer privacy handbook. Berkeley, CA: Peachpit Press.

Bronson, G.J. (2002). A First Book of Java<sup>TM</sup>. Pacific Groove, CA: Brooks/Cole Thomson Learning, Inc.

Darby, C., Griffin, J. & Haan, P. (2001). Beginning Java networking. Birmingham, UK: Wrox Press Ltd.

Data Encryption Standard. (1999). FIPS PUB 46-3 Data Encryption Standard (DES). Retrieved from http://csrc.nist.gov/publications/fips/fips46-3.pdf.

Deitel, H.M. & Deitel, P.J. (2001). Java<sup>TM</sup> How To Program. (4<sup>th</sup> ed.). Upper Saddle River, NJ: Prentice-Hall.

Deitel, H.M. (2002). Advance Java<sup>TM</sup> 2 Platform How To Program. Upper Saddle River, NJ: Prentice-Hall.

Fidanci, O.D., Diab, H., El-Ghazawi, T., Gaj, K. & Alexandridis, N. (2002). Implementation trade-offs of TripleDES in the SRC-6e Reconfigurable Computing Environment. MAPLD International Conference, The JohnHopkinsUniversity,Maryland.Retrievedhttp://klabs.org/richcontent/MAPLDCon02/papers/session d/d3 fidanci p.p df

Haney, J.D. (2006). The use of cryptography to create data file security: with the Rijndael cipher block. *Journal of Computing Sciences in College*. 21 (3), 30-39.

Henry, K. (2000). Getting started with PGP. Crossroads: The ACM magazine for students. 6 (5). doi:10.1145/345107.345119, http://dx.doi.org/10.1145/345107.345119.

Stallings, W. (2002). Network security essentials: application and standards. (2nd ed.). New Jersey: Prentice-Hall.

Table 1. Summary of PGP Services

Operation	Algorithms	Explanations	
Authentication/ Digital Signature	DSS/SHA or RSA/SHA	Hash code message was created using SHA-1. Message digests which have been encrypted using DSS or RSA with the sender's private key including the message.	
Confidentiality/ Message Encryption	CAST or IDEA or TDES with Diffie-Hellman or RSA	Message was encrypted using CAST-128 or IDEA or TDES with one-time session key. This key was encrypted by using Diffie-Hellman or RSA with the receiver's public key including message.	
Compression	ZIP	The message was compressed for storing or transmission purpose.	
Email compatibility	Radix 64	Enable the email to be access anyway and the message was encrypted and converts into ASCII using radix-64.	
Segmentation and Reassembly	-	Capability to hold the maximum capacity of the message size through segmentation.	

Source: Stallings (2002)











Figure 3. The Illustration of the Client/Server Concept



Figure 4. Sender/Receiver Architecture



Figure 5. The Example of Key



Figure 6. Running the Server

🖾 Command Prompt - java PGPftp		
Microsoft Windows 2000 [Version 5.00.2195] (C) Copyright 1985-2000 Microsoft Corp.		
C://cd_j2sdk1.4.0		
C:\j2sdk1.4.0>cd bin2		
C:\j2sdk1.4.0\bin2>java PGPftp 1. Connect to host 2. Wait for connections		
Please make a choice: 1		
CLIENT Please type in the host address: 127.0.0.1 Please type in the host port number: 8189	•	
	• //.	

Figure 7. Running the Client

🎘 SERVER PGP FTP		🚳 CLIENT PGP FTP	- 🗆 🗵
File:		File:	
	Browse		Browse
and a second			
encrypt decrypt		encrypt decrypt	
Send Reset Exit		Send Reset Exit	

Figure 8. The Interface for Client and Server



Figure 9. Decompression and Decryption Process



Figure 10. The Changes of the File Content in Relation with the Processes

# APPENDIX A

//CryptKey.java program, program to generate secret key //declaration import java.io.\*; import javax.crypto.\*; import javax.crypto.spec.SecretKeySpec; import java.security.spec.KeySpec; public class CryptKey { public static final String ALGORITHM = "DESede"; //using TDES algorithms public static final String KEY FILENAME = "key"; public static void main(String[] args) throws Exception { int keysize = Integer.parseInt(args[0]); //Create a secret key System.out.println("Generating key, size " + keysize + "..." ); KeyGenerator kg = KeyGenerator.getInstance(ALGORITHM); kg.init(keysize); SecretKey key = kg.generateKey(); System.out.println("Writing to disk "); OutputStream os = new FileOutputStream(KEY FILENAME); try {os.write(key.getEncoded()); } finally { try {os.close(); } catch(Exception e) { // do nothing  $\}\}$ public static SecretKey readSecretKey() throws Exception { //Read secret key System.out.println("Reading key...."); byte[] keyBuffer = new byte[(int) new File("key").length()]; InputStream is = new FileInputStream("key"); try {is.read(keyBuffer);} finally { try { is.close(); } catch(Exception e) { //do nothing } } //Use a generic key spec to represent the spec as an opaque key System.out.println("Creating opaque key... "); return new SecretKeySpec(keyBuffer, ALGORITHM);}}

#### APPENDIX B

//PGPftp.java program

//declaration

import java.io.\*;

import java.net.\*;

import java.awt.\*;
import java.awt.event.\*;

import javax.swing.\*;

import java.util.\*;

//for security

import java.security.spec.KeySpec;

import javax.crypto.\*;

//for compression

import java.util.zip.\*;

public class PGPftp extends Frame {

public static String hostAddress = "";

public static int portNumber = 0, maxClients = 0;

public static Vector sockets = null;

public static PGPftp tp;

public static String fileName = "",path = "";

public static int check = 0;

public static Socket connection = null;

public static ObjectOutputStream out = null;

public static ObjectInputStream in = null;

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

//read input from keyboard

InputStreamReader isr = new InputStreamReader(System.in);

BufferedReader stdin = new BufferedReader (isr);

//options for user System.out.println("1. Connect to host"); System.out.println("2. Wait for connections"); System.out.print("Please make a choice: "); System.out.flush(); int choice = Integer.parseInt(stdin.readLine()); if (choice == 1) { System.out.print("\nCLIENT....."); System.out.print("\nPlease type in the host address: "); System.out.flush(); hostAddress = stdin.readLine(); System.out.print("Please type in the host port number: ");

```
System.out.flush();
portNumber = Integer.parseInt(stdin.readLine());
                                                         }
if (choice == 2) {
System.out.print("\nSERVER.....");
System.out.print("\nGive the port number to listen for requests: ");
System.out.flush();
portNumber = Integer.parseInt(stdin.readLine());
System.out.print("Give the maximum number of clients for this server: ");
System.out.flush();
maxClients = Integer.parseInt(stdin.readLine());}
tp = new PGPftp(choice);}
public Label l;
public TextField tf;
public Button browse;
public Button send;
public Button reset;
public Button exit;
public Button encrypt;
public Button decrypt;
public PGPftp (int c) {
setTitle("PGP File Transfer");
setSize(300, 350);
setLayout(null);
addWindowListener(new WindowAdapter () { public void windowClosing (WindowEvent e)
{ System.exit(0); } } );
 l = new Label("File:");
add(1); 1.setBounds(15,87,50,20);
tf = new TextField("");
add(tf); tf.setBounds(13,114,200,20);
browse = new Button("Browse");
browse.addActionListener(new buttonListener());
add(browse); browse.setBounds(223,113,50,20);
encrypt = new Button("encrypt");
encrypt.addActionListener(new buttonListener());
add(encrypt); encrypt.setBounds(50,168,50,20);
decrypt = new Button("decrypt");
decrypt.addActionListener(new buttonListener());
add(decrypt); decrypt.setBounds(110,168,50,20);
send = new Button("Send");
send.addActionListener(new buttonListener());
```

add(send); send.setBounds(20,200,50,20); reset = new Button("Reset"); reset.addActionListener(new buttonListener()); add(reset); reset.setBounds(80,200,50,20); exit = new Button("Exit"); exit.addActionListener(new buttonListener()); add(exit); exit.setBounds(140,200,50,20); show();

//establish connection

if (c == 1) { check = 10; try {

connection = new Socket (hostAddress,portNumber);

out = new ObjectOutputStream(connection.getOutputStream());

out.flush();

in = new ObjectInputStream(connection.getInputStream());

int flag = 0; while (true) {

Object recieved = in.readObject();

switch (flag) {

case 0: if (recieved.equals("sot")) { flag++; } break;

case 1:

fileName = (String) recieved;

int option = JOptionPane.showConfirmDialog(this,connection.getInetAddress().getHostName()+" is sending you "+fileName+"!\nDo you want to recieve it?","Recieve Confirm",JOptionPane.YES\_NO\_OPTION, JOptionPane.QUESTION MESSAGE);

if (option == JOptionPane.YES\_OPTION) { flag++; } else { flag = 0; } break;

case 2:

byte[] b = (byte[])recieved;

FileOutputStream ff = new FileOutputStream(fileName);

ff.write(b);

flag = 0;

JOptionPane.showMessageDialog(this,"File Recieved!","Confirmation", JOptionPane.INFORMATION\_ MESSAGE); break; }

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

if (c == 2) {
sockets = new Vector();
check = 5;
try { ServerSocket connect = new ServerSocket(portNumber,maxClients);
while (true) {
sockets.addElement(new ThreadedSocket(connect.accept()));
Thread.yield();}

} catch (IOException ioe) {System.out.println(ioe);} } } public static String showDialog () { FileDialog fd = new FileDialog(new Frame(),"Select File...",FileDialog.LOAD); fd.show(); return fd.getDirectory()+fd.getFile();} private class buttonListener implements ActionListener //for button { public void actionPerformed (ActionEvent e) { byte[] array = null;

```
//button browse
if (e.getSource() == browse) {
    path = showDialog();
tf.setText(path);
int index = path.lastIndexOf("\\");
fileName = path.substring(index+1); }
```

```
//button encrypt
if(e.getSource() == encrypt){ try {
FileInputStream f = new FileInputStream (path);
int size = f.available();
array = new byte[size];
f.read(array,0,size);
```

```
//obtain input from keyboard
InputStreamReader isr = new InputStreamReader(System.in);
BufferedReader br = new BufferedReader(isr);
File file = new File(fileName);
long filelength = file.length();
FileInputStream fis = new FileInputStream(file);
DataInputStream dis = new DataInputStream(fis);
byte[] inputbuffer = new byte[(int)filelength];
SecretKey key = CryptKey.readSecretKey();
Cipher cipher = Cipher.getInstance(CryptKey.ALGORITHM);
cipher.init(Cipher.ENCRYPT_MODE, key);
```

```
//read from file
for(int i = 0; i<(int)filelength; i++) {
    inputbuffer[i] = dis.readByte();}
byte[] b1 = inputbuffer;
byte[] b2 = cipher.doFinal(b1);
System.out.print("\nEnter name for file encryption : "); //new name for encrypted file</pre>
```

String fileencrypt = br.readLine();

FileOutputStream fos1 = new FileOutputStream(fileencrypt);

DataOutputStream dos1 = new DataOutputStream(fos1);

//write into encrypted file for(int i = 0; i < b2.length; i++) { dos1.write(b2[i]); } System.out.print("\nEnter name for zip file : "); //new name for compressed file String zipfilename = br.readLine(); File zipfile = new File(zipfilename); File files = new File(fileencrypt); long fileslength = files.length(); FileInputStream fis1 = new FileInputStream (files); DataInputStream dis1 = new DataInputStream(fis1); FileOutputStream fos = new FileOutputStream(zipfile); ZipOutputStream zos = new ZipOutputStream(fos); ZipEntry nextzipentry = new ZipEntry(files.getName()); zos.putNextEntry(nextzipentry); try{ while(true){byte b = dis1.readByte(); zos.write(b); } }catch(EOFException eofe){ dis1.close(); }zos.close(); JOptionPane.showMessageDialog(null,"Process Encryption and Zip Success","Confirmation", JOptionPane. INFORMATION MESSAGE); } catch (Exception ex) {} } //button decrypt  $if(e.getSource() == decrypt) \{ try \}$ FileInputStream f = new FileInputStream (path); int size = f.available(); array = new byte[size]; f.read(array,0,size); InputStreamReader isr = new InputStreamReader(System.in); BufferedReader stdin = new BufferedReader(isr); ZipFile zipfile = new ZipFile(fileName); System.out.print("\nEnter name for unzip file : "); //insert new name for decompressed file String unzipfilename = stdin.readLine(); File unzipfile = new File(unzipfilename); Enumeration enumeration = zipfile.entries(); while (enumeration.hasMoreElements()){ ZipEntry nextzipentry = (ZipEntry) enumeration.nextElement(); InputStream is = zipfile.getInputStream(nextzipentry); DataInputStream dis = new DataInputStream(is); FileOutputStream fos = new FileOutputStream(unzipfile); DataOutputStream dos = new DataOutputStream (fos);

```
try {while(true) { fos.write(dis.readByte()); }
}catch(EOFException eofe) { }fos.close();}
```

//decompressed file to decrypted file
File file = new File(unzipfilename);
long filelength = file.length();
FileInputStream fis1 = new FileInputStream(file);
DataInputStream dis1 = new DataInputStream(fis1);
byte[] inputbuffer = new byte[(int)filelength];
SecretKey key = CryptKey.readSecretKey();
Cipher cipher = Cipher.getInstance(CryptKey.ALGORITHM);
cipher.init(Cipher.DECRYPT\_MODE, key);

```
// read from file
for(int i = 0; i<(int)filelength; i++){
inputbuffer[i]=dis1.readByte();}
byte[] b1 = inputbuffer;
byte[] b2 = cipher.doFinal(b1);
System.out.print("\nEnter name for file decryption : " );
String decfile = stdin.readLine();
FileOutputStream fos1 = new FileOutputStream(decfile);
DataOutputStream dos1 = new DataOutputStream(fos1);
for (int i = 0; i<b2.length; i++){dos1.write(b2[i]);}
System.out.println("Fail berjaya di decrypt ");
fos1.close();</pre>
```

JOptionPane.showMessageDialog(null,"Process Decryption and Unzip Success", "Confirmation", JOptionPane.INFORMATION\_MESSAGE);} catch (Exception ex) {}}

/button send if (e.getSource() == send) {try { FileInputStream f = new FileInputStream (path); int size = f.available(); array = new byte[size]; f.read(array,0,size);

```
//for server
if (check == 5) {
for (int i=0;i<sockets.size();i++) {
ThreadedSocket temp = (ThreadedSocket)sockets.elementAt(i);
temp.out.writeObject("sot");
temp.out.flush();
temp.out.writeObject(fileName);
temp.out.flush();</pre>
```

```
temp.out.writeObject(array);
temp.out.flush();
                       } }
//for client
if (check == 10) {
out.writeObject("sot");
out.flush();
out.writeObject(fileName);
out.flush();
out.writeObject(array);
out.flush(); }
JOptionPane.showMessageDialog(null,"File Sent!","Confirmation",JOptionPane.INFORMATION
                   } catch (Exception ex) {} }
MESSAGE);
//button reset
if (e.getSource() == reset) {
 tf.setText(""); }
//button exit
if(e.getSource() == exit){
System.exit(1); }
                     } }
                              }
class ThreadedSocket extends Thread {
public Socket socket;
public ObjectInputStream in;
public ObjectOutputStream out;
public ThreadedSocket (Socket s) {socket = s;
try {out = new ObjectOutputStream(socket.getOutputStream());
out.flush();
 in = new ObjectInputStream(socket.getInputStream());
} catch (Exception e) {System.out.println(e);} start(); }
public void run () {try {
 int flag = 0;
String fileName = "";
while (true) {
Object recieved = in.readObject();
switch (flag) {case 0:
if (recieved.equals("sot")) { flag++;
                                       } break;
```

```
case 1:
```

fileName = (String) recieved;

int option = JOptionPane.showConfirmDialog(null,socket.getInetAddress().getHostName()+ " is sending you

"+fileName+"!\nDo you want to recieve it?", "Receive Confirm", JOptionPane.YES NO OPTION, JOptionPane.QUESTION MESSAGE); if (option == JOptionPane.YES\_OPTION) { flag++; } else { flag = 0; } break; case 2: byte[] b = (byte[])recieved; FileOutputStream ff = new FileOutputStream(fileName); //ObjectOutputStream o = new ObjectOutpu // tStream(ff); //o.writeObject(b); //o.flush(); ff.write(b); flag = 0; //out.writeObject("rcn"); JOptionPane.showMessageDialog(null,"File Recieved!","Confirmation", JOptionPane.INFORMATION\_ MESSAGE); break; } Thread.yield();}} catch (Exception e) {System.out.println(e);} } }