SCHOOL OF INFORMATION TECHNOLOGY

CSE3502 – INFORMATION SECURITY MANAGEMENT

J – COMPONENT

TITLE: File Encryption and Decryption Using Password Based Encryption, MD5 & DES

SLOT: F2

FACULTY: IYAPPARAJA M

TEAM MEMBERS:

M Akhil – 17MIS0042

G Karunakar – 17MIS0432

N L N Ruthvik – 17MIS0376

V Harsha Vardhan – 17MIS0076

V Sesha Sai – 17MIS0072

ABSTRACT:

Password-Based Encryption gets the encryption key from the password. To make the task from keyword to key is very time-consuming for an attacker, most implementations of Password-based Encryption will be combined with a randomization system, known as salt. It should be that we want to effectively select an encryption key. We may want to encrypt files based on the passwords we enter, so we can remember them. In this case, the only information would be a password. Password-Based Encryption is used in the application because usually the attacker repeatedly tries to guess undetected keywords and is beyond the original sender / recipient control, if the keyword is used to log in to the server, it can detect many possibilities that are not properly done and in the worst case is to shut down the server to prevent more effort, if a tapper takes encrypted files that we use. Password Based Encryption with Message Digest (MD5) and Data Encryption Standard (DES) is a cryptographic method using algorithms that combine both hashing and standard encryption methods. MD5 was developed by Ronald Rivest where the MD5 takes messages of any length and generates a 128-bit message digest, and with the use of DES working in plaintext it is useful to return the same size ciphertext.

INTRODUCTION:

Technological developments are so rapid now, especially in the process of sending a message, but we often complain about safety. Security in data exchange is very important for users. This requires a mechanism for maintaining the authenticity of data in the exchange of data. As is the case in the data transmission many changes and the exchange of data made by irresponsible parties to take advantage. In sending messages through e-mail security is needed in order to maintain the integrity of the message.

For the example is hoax news case, where a person receives an electronic message from an incorrect news sender. It turns out that in the process of sending a message a change made by an irresponsible person who changed the message to sender incorrect information. In this case message security is necessary so that both parties do not feel disadvantaged, so the sender and the recipient must authenticate the message so that the two defendants can guarantee the source of the message.

Data security is always associated with cryptography. Cryptography is the study of how to maintain the security of a message and maintain the confidentiality of messages from irresponsibility by encoding into an

unreadable form (ciphertext). Cryptography studies mathematical techniques related to information security aspects such as confidentiality, data integrity and authentication. So, the authenticity of the data can be guaranteed. Cryptography studies mathematical techniques related to information security aspects such as confidentiality, data integrity and authentication. So, the authenticity of the data can be guaranteed.

OBJECTIVES:

To protect files from others to open it. We can encrypt files and store it in hard drive. Even if someone wanted to open it, s/he have to decrypt the file and then can open it. Since we encrypt the file using a password and also a password hash iteration is added, it is difficult for anybody to decrypt it.

PROPOSED IMPLEMENTATION:

The process that will be discussed in this project include two basic cryptography process are Encryption and Decryption with the same key is used for both processes. The use of the same key for both encryption and decryption process is also called Secret Key, Shared Key or Symmetric Key Cryptosystem. The Encryption process makes the message readable (plaintext) randomly unreadable (cipher text) then the Decryption process is the inverse of the encryption where this process will convert the cipher text into plaintext by using the same key. With the encryption and decryption process we will add the use of password-based encryption schemes in the protocol to implement symbolic and computational cryptography using symmetric, asymmetric, and password-based encryption. The usual alleged offline attacks can make consideration for the security process because the enemy makes every way to get any information about the passwords used. Therefore, we make no exceptions to the possibility that the enemy may play a role, with the possibility of installing a standard active attack, and obtaining data from interactions with other participants. It is useful to integrate these results with a password-based encryption analysis now.

PLATFORM:

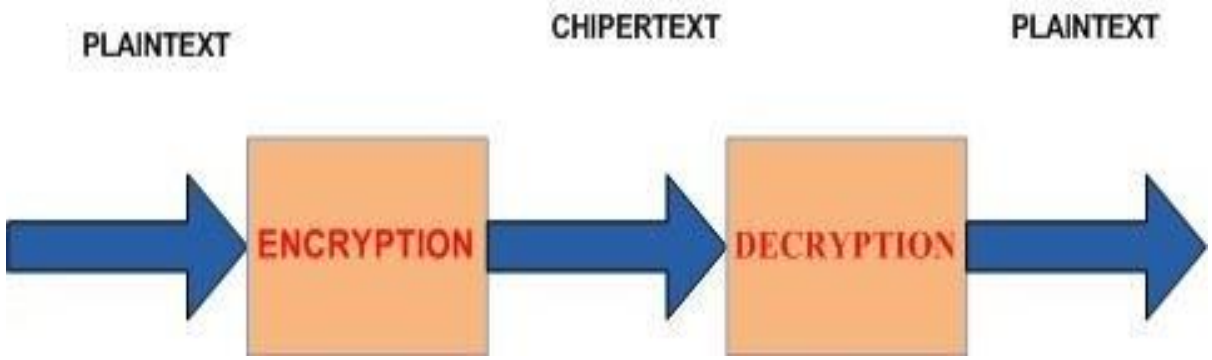
We implemented the following project in Eclipse and the code is written in Java language.

ALGORITHMS:

Cryptography - Cryptography is a data security technique to ensure data confidentiality, in addition to cryptographic understanding is the study of mathematical techniques related to information security such as data confidentiality, data validity, data integrity, data authentication.

Encryption and Decryption - Encryption is a process done to convert an undamaged message (plaintext) into an unreadable form (cipher text), decryption is a process done to convert an unreadable message into a readable and understandable form. The encryption and decryption process are governed by one or more cryptographic keys. Cryptosystem is a facility to convert plaintext to cipher text and vice versa. Based on the keys used for encryption and decryption, cryptography can be divided into symmetric key cryptography (Symmetric-key Cryptography) and asymmetric key cryptography (Asymmetric-key Cryptography).

Encryption and decryption process can be seen in the picture

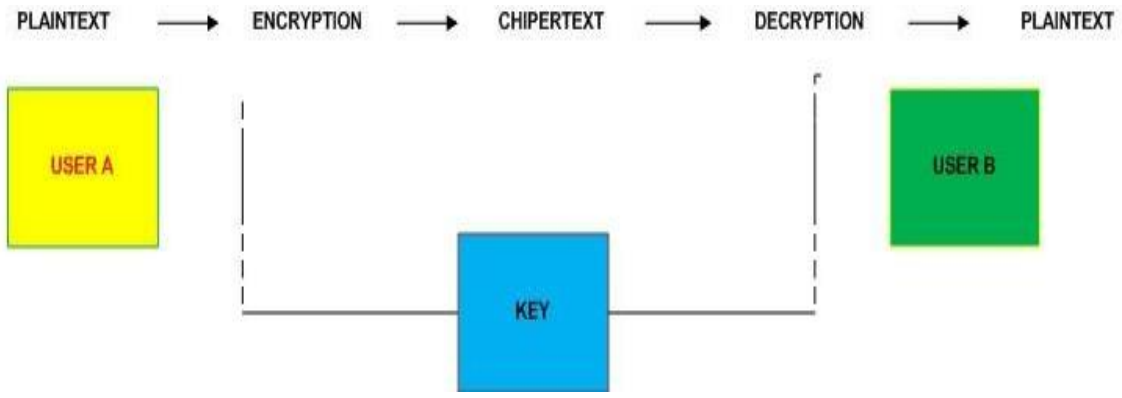


Cryptographic system or cryptosystem is a facility to convert plaintext to ciphertext and vice versa. In this system, the parameters that determine a particular involuntary transformation are called a set of keys. The encryption and decryption process are governed by one or more cryptographic keys. In general, the keys used for the process of encryption and description is not necessarily identical, depending on the system used. In general, the process of encryption and decryption operation can be explained mathematically as follows:

EK (M) = C (Encryption Process) DK (C) = M (Decryption Process)

In the message M we declare be message C by using the key K, while in the decryption process we use the key K and do the message C that has been in the encryption and generate the initial message that is M.

Symmetric Algorithms - Symmetric algorithms are cryptographic algorithm based on the key is divided into two namely the flow algorithm (Stream Cipher) and block algorithm (Block Cipher). In the Stream Cipher algorithm, the encoding process is oriented to one bit or one byte of data. While in the block algorithm, the encoding process is oriented to a set of bits or bytes of data. Examples of well-known symmetric algorithms are DES (Data Encryption Standard).



Symmetric Algorithms Process

This algorithm uses the same key for the encryption and decryption process. In a symmetric key cryptography system, the keys used for the encryption and decryption process are essentially identical, but one key can be derived from another key. These keys must be kept secret. Therefore, this system is often referred to as a secret key cipher system.

Asymmetric Algorithms - Asymmetric algorithms are algorithm that uses different keys for encryption and decryption. This algorithm is often called a public key algorithm because the key for encryption is made public (Public Key) or can be known by everyone, but the decryption key is only known by the person authorized to know the data is encoded or often called the private key

(Private Key). Examples of well-known algorithms using asymmetric keys are ECC and RSA. The process of asymmetric algorithm can be seen.

A simple public key encryption process involves the following four stages:

1. Each user in the network creates a pair of keys to use as an encryption and decryption key of the message to be received.

2. User then publishes its encryption key by placing its public key into a public place. Other key pairs are kept confidential.

3. If user A wants to send data to user B, it will encrypt the data by using the user's public key B.

4. When user B decrypts data from user A, it will use its own private key. No other party can decrypt the data because only B alone knows the private key B.

Comparison of Symmetric Algorithms with Asymmetric Algorithms

Both Symmetric and Asymmetric Algorithms have their own advantages and disadvantages.

Excess symmetric key cryptography:

1. The symmetric cryptography algorithm is designed so that the encryption and decryption process take a short time.

2. The size of the symmetric key is relatively short. A symmetric algorithm can be used to generate a random number.

3. A symmetric key algorithm can be constructed to produce a stronger cipher.

Authentication of direct data transmission is known from the received ciphertext, because the key is known only to the sender and the recipient only.

- Symmetric key cryptography shortcomings:
- Symmetric keys should be sent over a secure channel. Both communicating entities should maintain key secrecy.
- The key must be changed frequently, perhaps every communication.

Advantages of asymmetric key cryptography:

- Only private keys need to be kept confidential by every communicating entity. There is no need to send a private key as symmetric key cryptography.
- Pairs of public keys need not be changed, even in long periods of time.
- Can be used in the delivery of symmetric keys.
- Some public key algorithms can be used to digitally sign members on data.

Asymmetric key cryptography shortcomings

Data encryption and decryption are generally slower than symmetric cryptographic systems, because encryption and decryption use large numbers and involve large powers of operations.

- The size of ciphertext is bigger than plaintext.
- The key size is relatively larger than the symmetric key size.
- Because the public key is widely known and can be used by everyone, ciphertext does not provide information about authentication of the sender.

Attacks on Cryptography

Each attack in cryptography of a cryptanalyst seeks to find the key or find the plaintext of the ciphertext with the assumption of cryptanalysis knowing the cryptographic algorithm used. According to Kerckhoff's principle all cryptographic algorithms must be public, only secret keys.

The types of attacks in cryptography based on the attacker's involvement in communication:

- **Passive attack -** The attacker does not engage in direct communication with the sender and recipient and only wiretaps to obtain as much data or information as possible.
- **Active attack -** Attackers interfere with communication and influence the system to their advantage and attackers alter the flow of messages such as:
    - Delete a portion of ciphertext
    - Change the ciphertext
    - Inserting fake ciphertext
    - To reply the old messages

The types of attacks in cryptography are based on techniques used to find keys:

Exhaustive attack - Attackers use a way to find keys by trying all possible keys, surely manage to find the key if there is enough time.

Analytical attack - Attackers use the means by analysing the weaknesses of cryptographic algorithms to reduce the possibility of unlikely keys by solving mathematical equations. This method is usually faster to find the key than the exhaustive attack. A cryptographic algorithm is said to be safe when it meets three of the following criteria:

- The mathematical equations that describe the operation of cryptographic algorithms are so complex that the algorithm cannot be solved analytically.
- The cost to solve the ciphertext goes beyond the value of the information contained in the ciphertext.
- The time it takes to break the ciphertext beyond the length of time the information must be kept confidential.

METHODOLOGY:

Hash Function - The H hash function is a transform that takes input with size m and returns a fixed-size string called a hash value h (where, h = H (m)). This simple hash function has many different types of computing utility, but when used for cryptographic issues, hash functions are always added with a number of additional properties What is needed for a hash cryptography function:

1. Input with any length
2. The result has a fixed length output
3. H (x) is generally easy to calculate for any value of x
4. H (x) is one way
5. H (x) never has any problem with others

The H hash function is a one-way function because it is difficult to invert which means for the hash value h, it is difficult to find the input value x satisfying the equation H (x)= h. The value of a hash function declares a message or a longer document originating from the computation process. This is interesting because with the hash function, we can create a digital fingerprint for a document. The best-known examples of hash functions are MD2, MD5 and SHA. Perhaps a common use of hash cryptography is the creation of digital

signatures. Since hash functions are generally faster than other digital signature algorithms, hash functions are more commonly used to derive a hash value function by calculating a signature that results in a smaller hash value than the document itself. In addition, the public may provide a suggestion or opinion without disclosing the content of the opinions contained therein. This method is used in assigning dates to a document where by using hash functions, each person can assign date to the document without showing the contents of the document during the date allocation process.

Cryptographic hash function is a hash function that has some additional security properties that can be used for data security purposes. A hash function is a function that efficiently converts a finite input string to an output string with a fixed length called a hash value.

The characteristic of a cryptographic hash function:

- **Preimage resistant -** If it is known that hash value h is difficult (computationally not feasible) to get m where h = hash (m).
- **Second Preimage resistant -** If m1 input is known then it is difficult to find input m2 (not equal to m1) which causes hash (m1) = hash (m2)
- **Collision-resistant -** It is difficult to find two different inputs m1 and m2 causing hash (m1) = hash (m2).Some examples of cryptographic hash algorithms are MD4, MD5, SHA-0, SHA-1, SHA-256, SHA-512.

Encryption:

1. Plaintext is arranged into blocks m1, m2, ..., such that each block represents a value in the range 0 to p - 1.

2. Choose a random number k, in which case 0 □ k □ p – 1, such that k is relatively primed with p – 1.

3. Each block m is encrypted by the formula:

$$a = g^k \bmod p \quad b = y^{km} \bmod p$$

Set a and b are ciphertext for message blocks m. So, ciphertext size is twice the size of the plaintext.

Decryption:

To decrypt a and b secret key, x, and plaintext m are recovered by equation:

$$m = b/a^x \bmod p$$

because:

$$a^x \equiv g^{kx} \pmod{p}$$

then:

$$b/a^x \equiv y^k m/a^x \equiv g^{xk} m/g^{xk} \equiv m \pmod{p}$$

Which means that plaintext can be recovered from the ciphertex set a and b.

WORKING OF THE SYSTEM

Password Based Encryption -

Password Based Encryption (PBE) is a symmetric cryptographic method that uses a password-like key to perform the encryption process and uses the same key to perform the decryption process so that it will generate the same data as the original plain text data. Plain text data that has been encrypted will produce a cipher text that cannot be read by others. Cipher text is what will be sent to a second party so will have a reliable confidentiality. The resulting cipher text data will be changed according to the password data input provided. PBE cryptography is based on the hashing mechanism. A password and salt will be combined so that it will generate random data through the application function process and will be processed by the iteration count so that when the mixing process has finished it will produce the data in the form of cipher text.

PBE with MD5 and DES

PBE with MD5 and DES is a cryptographic method using the Message Digest 5 (MD5) and Data Encryption Standard (DES) algorithms. MD5 is the message digest algorithm

Developed by Ronald Rivest in 1991. MD5 takes messages of any length and generates a 128-bit message digest. On MD5 messages are processed in 512-bit blocks with four distinct rounds. DES, the acronym of the Data Encryption Standard, is the name of the Federal Information Processing Standard (FIPS), which describes the data encryption algorithm (DEA).

DES is also defined in ANSI standard X3.92. DEA is an improvement of the Lucifer algorithm developed by IBM in the early 1970s. Although the algorithm is essentially designed by IBM, the NSA and NBS (now NIST (National Institutes of Standards and Technology) play an important role in the

final stages of development. DEA, often called DES, has been extensively studied since its publication and is the best known and most widely used symmetric algorithm. DES has a 64-bit block size and uses a 56-bit key lock during execution (8-bit parity removed from a 64-bit key). When used for communication, both sender and receiver must know the same secret key, which can be used to encrypt and decrypt messages, or to generate and verify message authentication code (MAC).

MD5 (Message Digest 5)

Message Digest 5 (MD-5) is one of the most widely used of one-way hash functions. MD-5 is the fifth hash function designed by Ron Rivest. MD-5 is a development of MD-4 where there is an addition of one round. MD-5 processes the input text into 512-bit bits of blocks, divided into 32 bits of sub-blocks of 16 pieces. The output of the MD-5 is 4 blocks of 32 bits each which will be the usual 128 bits called the hash value.

MD5's main node has a 512-bit long message block that goes into 4 rounds. The output of MD-5 is 128 bits from the lowest byte A and the highest byte D Each message will be encrypted, first searched how many bits are contained in the message. We consider as much as b bits. Here b is an integer non-negative bit, b can be zero and not necessarily multiples of eight. Message Process in Block 16 Word.

In MD-5 there are also 4 (four) nonlinear functions each used in each operation (one function for one block), is:

$$F(X,Y,Z) = (X \wedge Y) \vee (\neg X \wedge Z)$$
$$G(X,Y,Z) = (X \wedge Y) \vee (Y \wedge \neg Z)$$
$$H(X,Y,Z) = X \oplus Y \oplus Z$$
$$I(X,Y,Z) = Y \oplus (X \vee \neg Z)$$

where $\wedge$ is logical and, $\vee$ is logical or and $\oplus$ is logical xor.

Here can be seen one operation of MD-5 with the operation used as an example is FF (a, b, c, d, Mj, s, ti) showing a = b + ((a + F (b, c, d) + Mj + ti) <<<s). If Mj represents the j- message of the sub-blocks (from 0 to 15) and <<<s describes the bit will be shifted to the left as much as s bit, then the fourth operation of each round is:

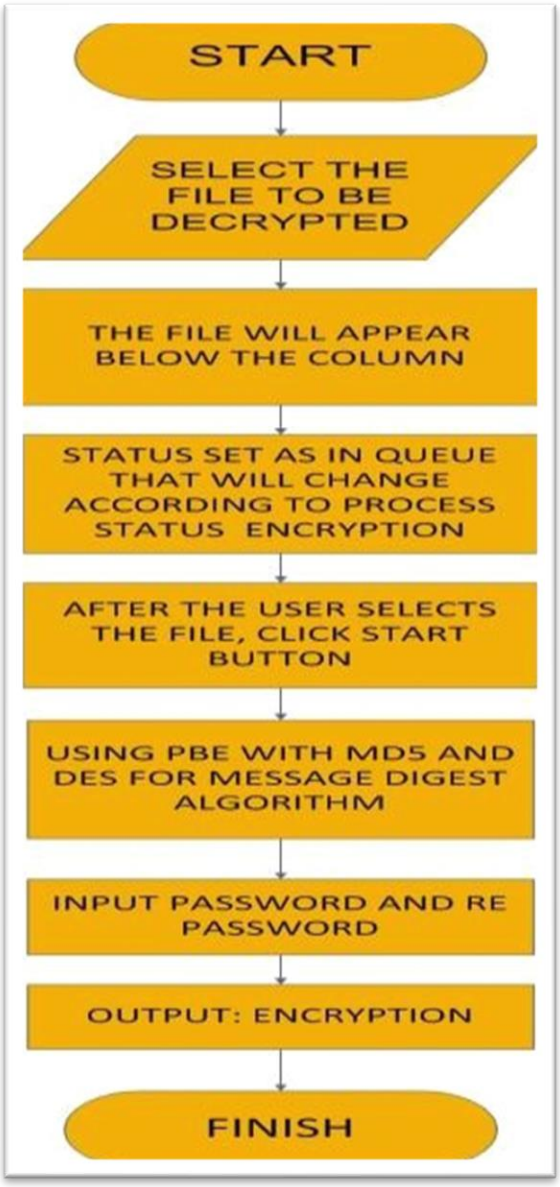
FF (a, b, c, d, Mj, s, ti) showing a = b + ((a + F (b, c, d) + Mj + ti) <<< s) GG (a, b, c, d, Mj, s, ti) showing a = b + ((a + G (b, c, d) + Mj + ti) <<< s) HH (a, b, c, d, Mj, s, ti) showing a = b +((a + H (b, c, d) + Mj + ti) <<< s) II (a, b, c, d, Mj, s, ti) showing a = b + ((a + I (b, c, d) + Mj + ti) <<< s)

DES (Data Encryption Standard)

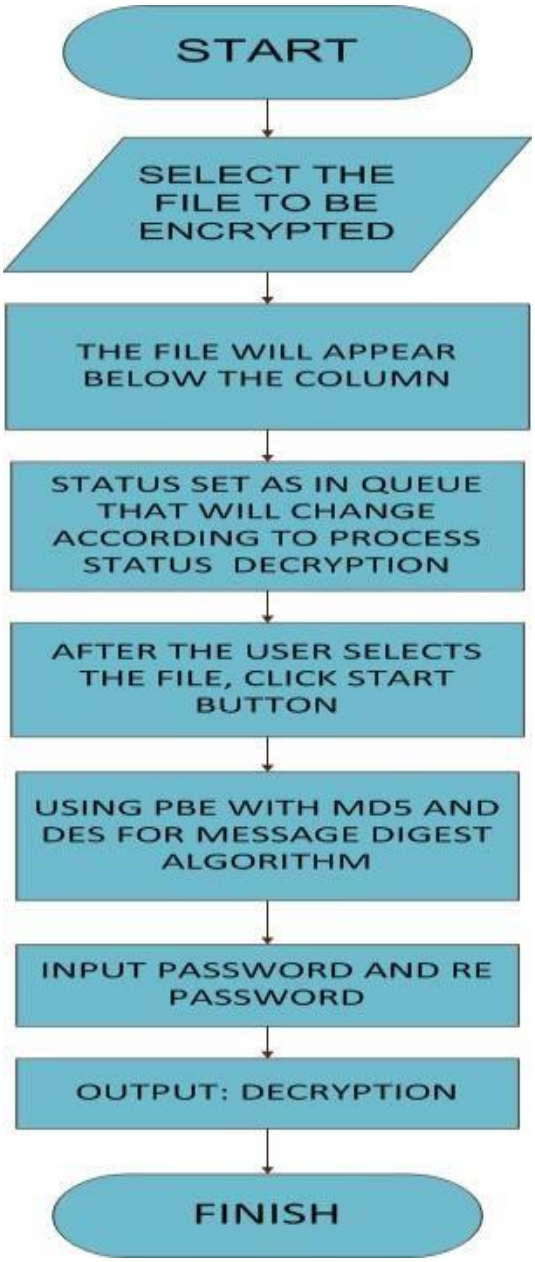
The most commonly used encryption scheme today is Data encryption Standard (DES). DES was adopted in 1977 by the National Bureau of Standards, now called the National Institute of Standards and Technology (NIST), as Federal Information Processing Standard 46 (FIPS PUB 46). In DES, data is encrypted inside 64-bit blocks using 56-bit keys. The DES algorithm converts 64-bit inputs in various steps to 64-bit outputs. The same step, with the same key, is used to decrypt the resulting ciphertext. In 1960, IBM started a project in computer cryptography led by Horst Feistel. The project was completed in 1971 with the development of an algorithm known as LUCIFER, which was sold to Lloyd's of London for use on the cash delivery system, which was also developed by IBM. LUCIFER is a block cipher that operates on 64-bit blocks, using a 128-bit key size. Due to the promising results, IBM later developed this system commercially. This effort is led by Walter Tuchman and Carl Meyer, and involves not only IBM, but also the technical consultants from the NSA. As a result, new versions of LUCIFER appear more resistant to cryptanalyst but by reducing key size to 56 bits so that it can be implemented on the system with a single processor. Meanwhile, the National Bureau of Standards (NBS) in 1973 issued a request for national cipher standards. IBM sent the results of the Tuchman-Meyer probe. This is the best algorithm proposed and adopted as Data Encryption Standard. DES works in bit models, or binary numbers 0 and 1. Each group of 4 bits forms a hexadecimal, or 16 based number. The binary number 0001 forms the hex numbers 1, and so on. DES works by encrypting each group consisting of 64 bits of data. To perform encryption, DES requires a key that also has a 64-bit size, but in practice the 8th bit of each group of 8 bits is ignored, so the key size becomes 56 bits. For example, if we want to encrypt the message "8787878787878787" with key "0E329232EA6D0D73", it will generate ciphertext "0000000000000000". If the ciphertext is decrypted using the same key, then the output is the original message. DES is a "block cipher", that is, DES works in plaintext with given size (64 bits) and returns ciphertext of the same size.

DESIGNING

In this application system encryption and encryption process with the method of PBE (password-based encryption) with MD5 and DES done on the structure of applications that have been made, can be seen from the flowchart chart as follows:



Flowchart Encryption



Flowchart Decryption

Implementation Code

```
package sumit.cryptbox.util;


import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.List;
import javax.swing.JOptionPane;
import javax.swing.JProgressBar;
import javax.swing.JTable;
import javax.swing.SwingWorker;
import javax.swing.table.TableModel;
import sumit.cryptbox.ui.dialogReEnter;


public class CryptAction extends SwingWorker<Integer, String>
{
   DateFormat objDateFormat = new SimpleDateFormat("dd/MM/yyyy HH:mm:ss");
   Date objDate = new Date();
   Utility objUtility = new Utility();
   String strMessageDigestAlgorithm;
   int intPasswordHashIteration;
   String strPassword;
   String strCryptAction;
   boolean boolOriginalFileDelete;
   String strCryptExtension;
   StringBuilder strBuilderMessage = new StringBuilder();
   JTable objJTable;
   TableModel objTableModel;
```

```
    JProgressBar objJProgressBar;
    public CryptAction(String strArgMessageDigestAlgorithm, int
intArgPasswordHashIteration, String strArgPassword, String strArgCryptAction, boolean
boolArgOriginalFileDelete, String strArgCryptExtension, StringBuilder strArgBuilderMessage,
JTable objArgJTable, JProgressBar objArgJProgressBar)
    {
        this.strMessageDigestAlgorithm = strArgMessageDigestAlgorithm;
        this.intPasswordHashIteration = intArgPasswordHashIteration;
        this.strPassword = strArgPassword;
        this.strCryptAction = strArgCryptAction;
        this.boolOriginalFileDelete = boolArgOriginalFileDelete;
        this.strCryptExtension = strArgCryptExtension;
        this.strBuilderMessage = strArgBuilderMessage;
        this.objJTable = objArgJTable;
        this.objTableModel = this.objJTable.getModel();
        this.objJProgressBar = objArgJProgressBar;
    }

    // This method is invoked when the worker is finished its task
    @Override
    protected void done()
    {
        try
        {
            // Get the number of matches. Note that the
            // method get will throw any exception thrown
            // during the execution of the worker.
            int intTotalFileCount = get();
            publish(objDateFormat.format(objDate) + "  " + "Cryptography action for " +
intTotalFileCount + " files are completed");
        }
```

```
        catch(Exception ex)
        {
            JOptionPane.showMessageDialog(null, ex, "CryptBox Error",
JOptionPane.ERROR_MESSAGE);
        }
    }
    @Override
    protected void process(List<String> chunks)
    {
        for(String message : chunks)
        {
            strBuilderMessage.append(message + "\n");
        }
    }
    @Override
    protected Integer doInBackground() throws Exception
    {
        int intTotalFileCount = objTableModel.getRowCount();
        try
        {
            for(int i=0; i<objTableModel.getRowCount(); i++)
            {
                if(objTableModel.getValueAt(i, 1).toString().compareTo("In Queue") == 0)
                {
                    int intCryptAction = 0;
                    int intDelete = 0;

                    objJTable.setValueAt("In progress", i, 1);
                    publish(objDateFormat.format(objDate) + "  " + "Cryptography action is in
progress for \"" + objTableModel.getValueAt(i, 0).toString() + "\"");
```

```
            String strSelectedFileExtension = objTableModel.getValueAt(i,
0).toString().substring(objTableModel.getValueAt(i, 0).toString().lastIndexOf('.') + 1);

            if(strCryptAction.compareTo("encrypt") == 0)

            {

               if(strSelectedFileExtension.compareTo(strCryptExtension) == 0)

               {

                  int intUserOption = JOptionPane.showConfirmDialog(null, "Are you sure to
encrypt an encrypted file?\n\nYES - Encrypt an encrypted file\nNO - Decrypt an encrypted
file\nCANCEL - Cancel encryption of an encrypted file", "CryptBox",
JOptionPane.YES_NO_CANCEL_OPTION, JOptionPane.INFORMATION_MESSAGE);

                  if(intUserOption == JOptionPane.YES_OPTION)

                  {

                     EncryptAction(objTableModel.getValueAt(i, 0).toString(),
strMessageDigestAlgorithm, intPasswordHashIteration, strPassword, i);

                  }

                  else if(intUserOption == JOptionPane.NO_OPTION)

                  {

                     DecryptAction(objTableModel.getValueAt(i, 0).toString(),
strMessageDigestAlgorithm, intPasswordHashIteration, strPassword, i);

                  }

                  else if(intUserOption == JOptionPane.CANCEL_OPTION)

                  {

                     objJTable.setValueAt("Encrypting failed", i, 1);

                     objJTable.setValueAt("Encrypting file was terminated by user", i, 2);

                     publish(objDateFormat.format(objDate) + "  " + "Encrypting file was
terminated by user");

                  }

               }

               else

               {

                  EncryptAction(objTableModel.getValueAt(i, 0).toString(),
strMessageDigestAlgorithm, intPasswordHashIteration, strPassword, i);
```

```
                }
            }
            if(strCryptAction.compareTo("decrypt") == 0)
            {
                if(strSelectedFileExtension.compareTo(strCryptExtension) != 0)
                {
                    int intUserOption = JOptionPane.showConfirmDialog(null, "Are you sure to
decrypt a normal file?\n\nYES - Decrypt a normal file\nNO - Encrypt a normal file\nCANCEL -
Cancel decryption of a normal file", "CryptBox", JOptionPane.YES_NO_CANCEL_OPTION,
JOptionPane.INFORMATION_MESSAGE);
                    if(intUserOption == JOptionPane.YES_OPTION)
                    {
                        DecryptAction(objTableModel.getValueAt(i, 0).toString(),
strMessageDigestAlgorithm, intPasswordHashIteration, strPassword, i);
                    }
                    else if(intUserOption == JOptionPane.NO_OPTION)
                    {
                        EncryptAction(objTableModel.getValueAt(i, 0).toString(),
strMessageDigestAlgorithm, intPasswordHashIteration, strPassword, i);
                    }
                    else if(intUserOption == JOptionPane.CANCEL_OPTION)
                    {
                        objJTable.setValueAt("Decrypting failed", i, 1);
                        objJTable.setValueAt("Decrypting file was terminated by user", i, 2);
                        publish(objDateFormat.format(objDate) + "  " + "Decrypting file was
terminated by user");
                    }
                }
                else
                {
                    DecryptAction(objTableModel.getValueAt(i, 0).toString(),
strMessageDigestAlgorithm, intPasswordHashIteration, strPassword, i);
```

```
                }
              }
            }
            else
            {
              objJTable.setValueAt("No action taken", i, 1);
              objJTable.setValueAt("The file is not in queue", i, 2);
              publish(objDateFormat.format(objDate) + "  " + "No action taken because the file
is not in queue");
            }
            // update the progress
            objJProgressBar.setValue((i+1) * 100 / intTotalFileCount);
            publish(objDateFormat.format(objDate) + "  " + "----------------------------------------------
-");
          }
        }
        catch(Exception ex)
        {
          JOptionPane.showMessageDialog(null, ex, "CryptBox Error",
JOptionPane.ERROR_MESSAGE);
        }
        return intTotalFileCount;
    }
    private void EncryptAction(String strFileName, String strInterMessageDigestAlgorithm, int
intInterPasswordHashIteration, String strInterPassword, int intTableRowNo)
    {
        int intCryptAction = 1;
        int intDelete = 1;
        try
        {
          do
```

```
        {
            objJTable.setValueAt("Encrypting file", intTableRowNo, 2);
            publish(objDateFormat.format(objDate) + "  " + "Encrypting file is in progress");
            intCryptAction = objUtility.EncryptFile(strFileName,
strInterMessageDigestAlgorithm, intInterPasswordHashIteration, strInterPassword,
strCryptExtension);
            if(intCryptAction == 0)
            {
               objJTable.setValueAt("Encrypting file is successful", intTableRowNo, 2);
               publish(objDateFormat.format(objDate) + "  " + "Encrypting file is successful");
               if(boolOriginalFileDelete == true)
               {
                  objJTable.setValueAt("Deleting file", intTableRowNo, 2);
                  publish(objDateFormat.format(objDate) + "  " + "Deleting file is in progress");
                  intDelete = objUtility.DeleteFile(strFileName);
                  if(intDelete == 0)
                  {
                     objJTable.setValueAt("Encryption completed successfully", intTableRowNo,
1);
                     objJTable.setValueAt("Deleting file is successful", intTableRowNo, 2);
                     publish(objDateFormat.format(objDate) + "  " + "Encryption completed
successfully and deleting file is successful");
                  }
                  else
                  {
                     objJTable.setValueAt("Encryption completed successfully with error",
intTableRowNo, 1);
                     objJTable.setValueAt("Deleting file has failed", intTableRowNo, 2);
                     publish(objDateFormat.format(objDate) + "  " + "Encryption completed
successfully with error because deleting file has failed");
                  }
               }
```

```
            else
            {
               objJTable.setValueAt("Encryption completed successfully", intTableRowNo, 1);
               objJTable.setValueAt("File is not deleted", intTableRowNo, 2);
               publish(objDateFormat.format(objDate) + "  " + "Encryption completed
successfully and file is not deleted");
            }
         }
         else
         {
            objJTable.setValueAt("Encrypting file has failed. Waiting for user response",
intTableRowNo, 2);
            publish(objDateFormat.format(objDate) + "  " + "Encrypting file has failed.
Waiting for user response");
            dialogReEnter objDialogReEnter = new dialogReEnter();
            objDialogReEnter.setVisible(true);
            if(objDialogReEnter.boolReEnter == true)
            {
               strInterMessageDigestAlgorithm =
objDialogReEnter.strMessageDigestAlgorithm;
               intInterPasswordHashIteration = objDialogReEnter.intPasswordHashIteration;
               strInterPassword = objDialogReEnter.strPassword;
            }
            else
            {
               objJTable.setValueAt("Encryption failed", intTableRowNo, 1);
               objJTable.setValueAt("Encrypting file was terminated by user",
intTableRowNo, 2);
               publish(objDateFormat.format(objDate) + "  " + "Encrypting file was
terminated by user");
               intCryptAction = 0;
            }
```

```
            }
          }while(intCryptAction == 1);
        }
        catch(Exception ex)
        {
          JOptionPane.showMessageDialog(null, ex, "CryptBox Error",
JOptionPane.ERROR_MESSAGE);
          try
          {
            objUtility.DeleteFile(strFileName + "." + strCryptExtension);
          }
          catch(Exception ex1)
          {
            JOptionPane.showMessageDialog(null, ex1, "CryptBox Error",
JOptionPane.ERROR_MESSAGE);
          }
        }
      }
    private void DecryptAction(String strFileName, String strInterMessageDigestAlgorithm, int
intInterPasswordHashIteration, String strInterPassword, int intTableRowNo)
    {
      int intCryptAction = 1;
      int intDelete = 1;
      try
      {
        do
        {
          objJTable.setValueAt("Decrypting file", intTableRowNo, 2);
          publish(objDateFormat.format(objDate) + "  " + "Decrypting file is in progress");
```

```
            intCryptAction = objUtility.DecryptFile(strFileName,
strInterMessageDigestAlgorithm, intInterPasswordHashIteration, strInterPassword,
strCryptExtension);
            if(intCryptAction == 0)
            {
               objJTable.setValueAt("Decrypting file is successful", intTableRowNo, 2);
               publish(objDateFormat.format(objDate) + "  " + "Decrypting file is successful");
               if(boolOriginalFileDelete == true)
               {
                  objJTable.setValueAt("Deleting file", intTableRowNo, 2);
                  publish(objDateFormat.format(objDate) + "  " + "Deleting file is in progress");
                  intDelete = objUtility.DeleteFile(strFileName);
                  if(intDelete == 0)
                  {
                     objJTable.setValueAt("Decryption completed successfully", intTableRowNo,
1);
                     objJTable.setValueAt("Deleting file is successful", intTableRowNo, 2);
                     publish(objDateFormat.format(objDate) + "  " + "Decryption completed
successfully and deleting file is successful");
                  }
                  else
                  {
                     objJTable.setValueAt("Decryption completed successfully with error",
intTableRowNo, 1);
                     objJTable.setValueAt("Deleting file has failed", intTableRowNo, 2);
                     publish(objDateFormat.format(objDate) + "  " + "Decryption completed
successfully with error because deleting file has failed");
                  }
               }
               else
               {
                  objJTable.setValueAt("Decryption completed successfully", intTableRowNo, 1);
```

```
                objJTable.setValueAt("File is not deleted", intTableRowNo, 2);
                publish(objDateFormat.format(objDate) + "  " + "Decryption completed
successfully and file is not deleted");
              }
            }
            else
            {
              objJTable.setValueAt("Decrypting file has failed. Waiting for user response",
intTableRowNo, 2);
              publish(objDateFormat.format(objDate) + "  " + "Decrypting file has failed.
Waiting for user response");
              dialogReEnter objDialogReEnter = new dialogReEnter();
              objDialogReEnter.setVisible(true);
              if(objDialogReEnter.boolReEnter == true)
              {
                strInterMessageDigestAlgorithm =
objDialogReEnter.strMessageDigestAlgorithm;
                intInterPasswordHashIteration = objDialogReEnter.intPasswordHashIteration;
                strInterPassword = objDialogReEnter.strPassword;
              }
              else
              {
                objJTable.setValueAt("Decryption failed", intTableRowNo, 1);
                objJTable.setValueAt("Decrypting file was terminated by user",
intTableRowNo, 2);
                publish(objDateFormat.format(objDate) + "  " + "Decrypting file was
terminated by user");
                intCryptAction = 0;
              }
            }
          }while(intCryptAction == 1);
        }
```

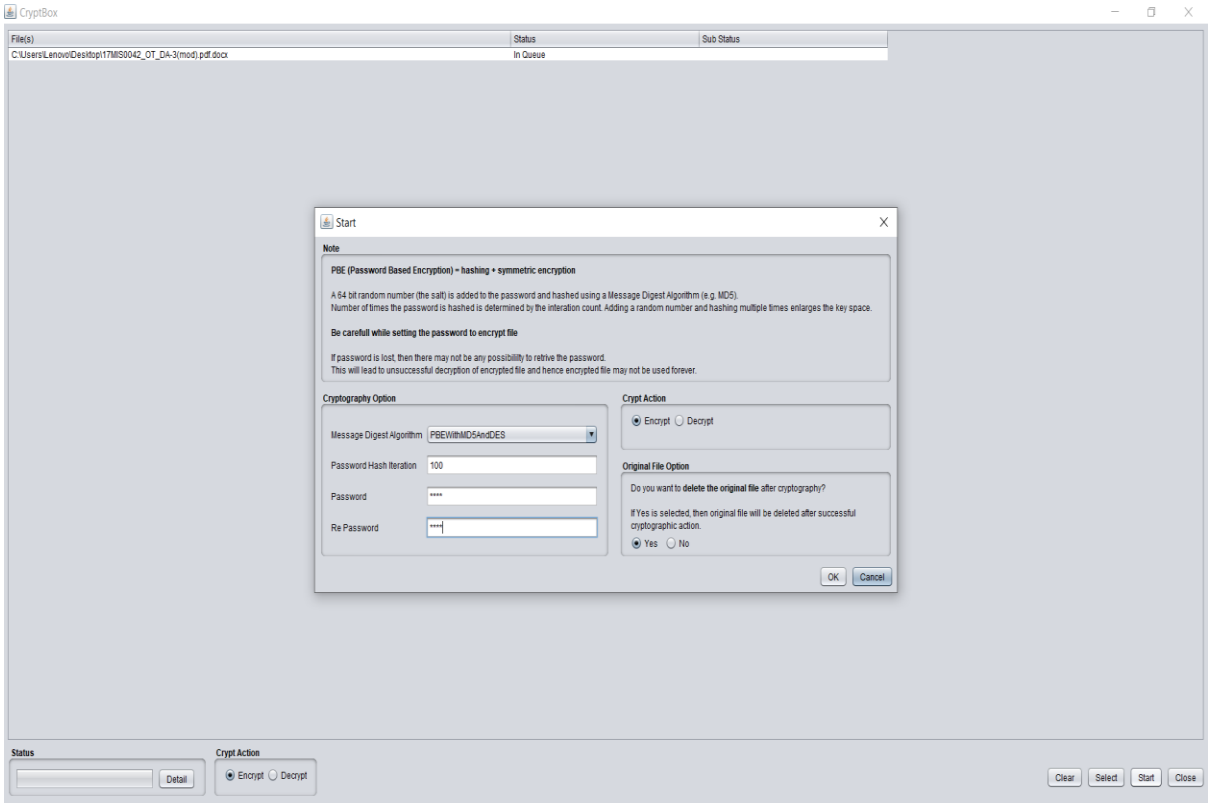
```
    catch(Exception ex)
    {
      JOptionPane.showMessageDialog(null, ex, "CryptBox Error",
JOptionPane.ERROR_MESSAGE);
      try
      {
        objUtility.DeleteFile(strFileName.substring(0, strFileName.length() -
strCryptExtension.length() - 1));
      }
      catch(Exception ex1)
      {
        JOptionPane.showMessageDialog(null, ex1, "CryptBox Error",
JOptionPane.ERROR_MESSAGE);
      }
    }
  }
}
```

RESULTS AND DISCUSSION

We must protect the files of others who will open them for bad interests. That's when I thought about encrypting my files and storing them on my hard drive. Even if someone wants to open it he should decrypt the file and then be able to open it.

Since I encrypt files using passwords and also hashes added, it's hard for anyone to decrypt them. Files can be added in the app to encrypt or decrypt in two ways. Either click the select button, choose the encryption key and will help the user to choose which files will be encrypted. Select the decryption key to help the user to select the encryption file. The application will generate an encryption and decryption file depending on which button will be selected. The app uses PBE with MD5, to enter the default Iteration hash password 100. How many times the hash password is determined by the iteration count. Enter a password to protect the file and be used for cryptography. Resetting a password must be exactly the same as the initial password to check if the password is the same as you want. When decrypting an encrypted file, the user needs the same Message Digest Algorithm, Hash Iteration and password used to encrypt the file. If the user provides an incorrect Message Digest Algorithm, Password Hash Iteration or Password when decrypting a file that does not match the Encrypted file, the application will prompt to re-enter the details, so that it can be correctly decrypted. At the moment Encryption password has to be provided for a file must be same for the file at the time of decryption. Encryption and decryption take time depending on the size of the file to be processed. For example, if the file is 2 GB, it takes about 20 minutes to encrypt or decrypt the file.

Applications can be viewed with the following picture:



CONCLUSION:

Applications that have been designed can generate encryption or decryption files, by obtaining the encryption key from the password. The technique generates the secret key of an artificial password called Password-Based Encryption and by using MD5 because it uses an algorithm that combines standard hashing and encryption methods as well as DES that works in plaintext useful for returning the same size cipher text. Password-Based Encryption = encryption hashing+ 64-bit symmetric randomly added to password and hash using MD5.

REFERENCES:

[1] Munir, Rinaldi., 2006, Kriptografi, Bandung, Informatika.

[2] Martin Abadi, Bogdan Warinschi, Password-Based Encryption Analyzed, Computer Science University of California, Standford University.

[3] Singh, S Preet and Maini Raman. "Comparison of Data Encryption Algorithm", International journal of Computer Science and Communication, vol.2, No.1, january-June, pp. 125-127.

[4] Stallings W., 1999, Cryptography and Network Security Principles and Practice second edition. Prentice Hall,New Jersey, USA

[5] Stinson, Cryptography, Theory and Practice:; CRC Press; Second edition; 2000.