LAMPIRAN

Enkripsi.java

```
import java.awt.FileDialog;
import java.awt.event.ActionEvent;
import java.io.BufferedInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.security.Key;
import javax.crypto.BadPaddingException;
import javax.crypto.Cipher;
import javax.crypto.IllegalBlockSizeException;
import javax.crypto.SecretKeyFactory;
import javax.crypto.ShortBufferException;
import javax.crypto.spec.DESKeySpec;
import javax.swing.JFrame;
import javax.swing.JOptionPane;
import javax.swing.JPasswordField;
import javax.swing.ProgressMonitor;
import javax.swing.ProgressMonitorInputStream;
```

```
import java.security.GeneralSecurityException;
import java.security.InvalidKeyException;
import java.io.BufferedOutputStream;
import java.util.Arrays;
import java.util.Enumeration;
import java.net.NetworkInterface;
import java.io.UnsupportedEncodingException;
import java.security.GeneralSecurityException;
import java.security.SecureRandom;
import java.security.MessageDigest;
import javax.crypto.spec.SecretKeySpec;
import javax.crypto.Mac;
import javax.crypto.spec.IvParameterSpec;

import java.io.*;
import java.util.Properties;/**
 * Title:
 *
 * Description:
 *
 * Copyright: Copyright (c) 2008
 *
 * Company:
 *
 * @author not attributable
 * @version 1.0
 */
public class Enkripsi {
```

```
 static int Nb=4;                          // number of columns in the state &
expanded key
 static int Nr =10;                        // number of rounds in encryption
 static int Nk= 4;                         // number of columns in a key

 static  char  Sbox[] = {                  // forward s-box
 0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x01, 0x67, 0x2b, 0xfe,
0xd7, 0xab, 0x76,
 0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0, 0xad, 0xd4, 0xa2, 0xaf, 0x9c,
0xa4, 0x72, 0xc0,
 0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc, 0x34, 0xa5, 0xe5, 0xf1, 0x71,
0xd8, 0x31, 0x15,
 0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a, 0x07, 0x12, 0x80, 0xe2, 0xeb,
0x27, 0xb2, 0x75,
 0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0, 0x52, 0x3b, 0xd6, 0xb3, 0x29,
0xe3, 0x2f, 0x84,
 0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b, 0x6a, 0xcb, 0xbe, 0x39, 0x4a,
0x4c, 0x58, 0xcf,
 0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85, 0x45, 0xf9, 0x02, 0x7f, 0x50,
0x3c, 0x9f, 0xa8,
 0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5, 0xbc, 0xb6, 0xda, 0x21, 0x10,
0xff, 0xf3, 0xd2,
 0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17, 0xc4, 0xa7, 0x7e, 0x3d, 0x64,
0x5d, 0x19, 0x73,
 0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88, 0x46, 0xee, 0xb8, 0x14, 0xde,
0x5e, 0x0b, 0xdb,
 0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c, 0xc2, 0xd3, 0xac, 0x62, 0x91,
0x95, 0xe4, 0x79,
```

```
 0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9, 0x6c, 0x56, 0xf4, 0xea, 0x65,
0x7a, 0xae, 0x08,
 0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6, 0xe8, 0xdd, 0x74, 0x1f, 0x4b,
0xbd, 0x8b, 0x8a,
 0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e, 0x61, 0x35, 0x57, 0xb9, 0x86,
0xc1, 0x1d, 0x9e,
 0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94, 0x9b, 0x1e, 0x87, 0xe9, 0xce,
0x55, 0x28, 0xdf,
 0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68, 0x41, 0x99, 0x2d, 0x0f, 0xb0,
0x54, 0xbb, 0x16
 };

// combined Xtimes2[Sbox[]]
  static char Xtime2Sbox[] = {
 0xc6, 0xf8, 0xee, 0xf6, 0xff, 0xd6, 0xde, 0x91, 0x60, 0x02, 0xce, 0x56, 0xe7,
0xb5, 0x4d, 0xec,
 0x8f, 0x1f, 0x89, 0xfa, 0xef, 0xb2, 0x8e, 0xfb, 0x41, 0xb3, 0x5f, 0x45, 0x23,
0x53, 0xe4, 0x9b,
 0x75, 0xe1, 0x3d, 0x4c, 0x6c, 0x7e, 0xf5, 0x83, 0x68, 0x51, 0xd1, 0xf9, 0xe2,
0xab, 0x62, 0x2a,
 0x08, 0x95, 0x46, 0x9d, 0x30, 0x37, 0x0a, 0x2f, 0x0e, 0x24, 0x1b, 0xdf, 0xcd,
0x4e, 0x7f, 0xea,
 0x12, 0x1d, 0x58, 0x34, 0x36, 0xdc, 0xb4, 0x5b, 0xa4, 0x76, 0xb7, 0x7d, 0x52,
0xdd, 0x5e, 0x13,
 0xa6, 0xb9, 0x00, 0xc1, 0x40, 0xe3, 0x79, 0xb6, 0xd4, 0x8d, 0x67, 0x72, 0x94,
0x98, 0xb0, 0x85,
 0xbb, 0xc5, 0x4f, 0xed, 0x86, 0x9a, 0x66, 0x11, 0x8a, 0xe9, 0x04, 0xfe, 0xa0,
0x78, 0x25, 0x4b,
```

```
 0xa2, 0x5d, 0x80, 0x05, 0x3f, 0x21, 0x70, 0xf1, 0x63, 0x77, 0xaf, 0x42, 0x20,
0xe5, 0xfd, 0xbf,
 0x81, 0x18, 0x26, 0xc3, 0xbe, 0x35, 0x88, 0x2e, 0x93, 0x55, 0xfc, 0x7a, 0xc8,
0xba, 0x32, 0xe6,
 0xc0, 0x19, 0x9e, 0xa3, 0x44, 0x54, 0x3b, 0x0b, 0x8c, 0xc7, 0x6b, 0x28, 0xa7,
0xbc, 0x16, 0xad,
 0xdb, 0x64, 0x74, 0x14, 0x92, 0x0c, 0x48, 0xb8, 0x9f, 0xbd, 0x43, 0xc4, 0x39,
0x31, 0xd3, 0xf2,
 0xd5, 0x8b, 0x6e, 0xda, 0x01, 0xb1, 0x9c, 0x49, 0xd8, 0xac, 0xf3, 0xcf, 0xca,
0xf4, 0x47, 0x10,
 0x6f, 0xf0, 0x4a, 0x5c, 0x38, 0x57, 0x73, 0x97, 0xcb, 0xa1, 0xe8, 0x3e, 0x96,
0x61, 0x0d, 0x0f,
 0xe0, 0x7c, 0x71, 0xcc, 0x90, 0x06, 0xf7, 0x1c, 0xc2, 0x6a, 0xae, 0x69, 0x17,
0x99, 0x3a, 0x27,
 0xd9, 0xeb, 0x2b, 0x22, 0xd2, 0xa9, 0x07, 0x33, 0x2d, 0x3c, 0x15, 0xc9, 0x87,
0xaa, 0x50, 0xa5,
 0x03, 0x59, 0x09, 0x1a, 0x65, 0xd7, 0x84, 0xd0, 0x82, 0x29, 0x5a, 0x1e, 0x7b,
0xa8, 0x6d, 0x2c
 };

// combined Xtimes3[Sbox[]]
  static char Xtime3Sbox[] = {
 0xa5, 0x84, 0x99, 0x8d, 0x0d, 0xbd, 0xb1, 0x54, 0x50, 0x03, 0xa9, 0x7d, 0x19,
0x62, 0xe6, 0x9a,
 0x45, 0x9d, 0x40, 0x87, 0x15, 0xeb, 0xc9, 0x0b, 0xec, 0x67, 0xfd, 0xea, 0xbf,
0xf7, 0x96, 0x5b,
 0xc2, 0x1c, 0xae, 0x6a, 0x5a, 0x41, 0x02, 0x4f, 0x5c, 0xf4, 0x34, 0x08, 0x93,
0x73, 0x53, 0x3f,
```

0x0c, 0x52, 0x65, 0x5e, 0x28, 0xa1, 0x0f, 0xb5, 0x09, 0x36, 0x9b, 0x3d, 0x26, 0x69, 0xcd, 0x9f,
0x1b, 0x9e, 0x74, 0x2e, 0x2d, 0xb2, 0xee, 0xfb, 0xf6, 0x4d, 0x61, 0xce, 0x7b, 0x3e, 0x71, 0x97,
0xf5, 0x68, 0x00, 0x2c, 0x60, 0x1f, 0xc8, 0xed, 0xbe, 0x46, 0xd9, 0x4b, 0xde, 0xd4, 0xe8, 0x4a,
0x6b, 0x2a, 0xe5, 0x16, 0xc5, 0xd7, 0x55, 0x94, 0xcf, 0x10, 0x06, 0x81, 0xf0, 0x44, 0xba, 0xe3,
0xf3, 0xfe, 0xc0, 0x8a, 0xad, 0xbc, 0x48, 0x04, 0xdf, 0xc1, 0x75, 0x63, 0x30, 0x1a, 0x0e, 0x6d,
0x4c, 0x14, 0x35, 0x2f, 0xe1, 0xa2, 0xcc, 0x39, 0x57, 0xf2, 0x82, 0x47, 0xac, 0xe7, 0x2b, 0x95,
0xa0, 0x98, 0xd1, 0x7f, 0x66, 0x7e, 0xab, 0x83, 0xca, 0x29, 0xd3, 0x3c, 0x79, 0xe2, 0x1d, 0x76,
0x3b, 0x56, 0x4e, 0x1e, 0xdb, 0x0a, 0x6c, 0xe4, 0x5d, 0x6e, 0xef, 0xa6, 0xa8, 0xa4, 0x37, 0x8b,
0x32, 0x43, 0x59, 0xb7, 0x8c, 0x64, 0xd2, 0xe0, 0xb4, 0xfa, 0x07, 0x25, 0xaf, 0x8e, 0xe9, 0x18,
0xd5, 0x88, 0x6f, 0x72, 0x24, 0xf1, 0xc7, 0x51, 0x23, 0x7c, 0x9c, 0x21, 0xdd, 0xdc, 0x86, 0x85,
0x90, 0x42, 0xc4, 0xaa, 0xd8, 0x05, 0x01, 0x12, 0xa3, 0x5f, 0xf9, 0xd0, 0x91, 0x58, 0x27, 0xb9,
0x38, 0x13, 0xb3, 0x33, 0xbb, 0x70, 0x89, 0xa7, 0xb6, 0x22, 0x92, 0x20, 0x49, 0xff, 0x78, 0x7a,
0x8f, 0xf8, 0x80, 0x17, 0xda, 0x31, 0xc6, 0xb8, 0xc3, 0xb0, 0x77, 0x11, 0xcb, 0xfc, 0xd6, 0x3a
};

```
// exchanges columns in each of 4 rows
// row0 - unchanged,
//row1- shifted left 1,
// row2 - shifted left 2
//and row3 - shifted left 3

 void ShiftRows (char state[])
 {
 char tmp;

        // just substitute row 0
       try {
          state[0] = Sbox[state[0] % 256];
          state[4] = Sbox[state[4] % 256];
          state[8] = Sbox[state[8] % 256];
          state[12] = Sbox[state[12] % 256];
       } catch (Exception ex) {
          ex.printStackTrace();
       }

       try {
          // rotate row 1
          tmp = Sbox[state[1] % 256];
          state[1] = Sbox[state[5] % 256];
          state[5] = Sbox[state[9] % 256];
          state[9] = Sbox[state[13] % 256];
          state[13] = tmp;
       } catch (Exception ex1) {
          ex1.printStackTrace();
```

```
        }

        try {
            // rotate row 2
            tmp = Sbox[state[2] % 256];
            state[2] = Sbox[state[10] % 256];
            state[10] = tmp;
            tmp = Sbox[state[6] % 256];
            state[6] = Sbox[state[14] % 256];
            state[14] = tmp;
        } catch (Exception ex2) {
            ex2.printStackTrace();
        }

        try {
            // rotate row 3
            tmp = Sbox[state[15] % 256];
            state[15] = Sbox[state[11] % 256];
            state[11] = Sbox[state[7] % 256];
            state[7] = Sbox[state[3] % 256];
            state[3] = tmp;
        } catch (Exception ex3) {
            ex3.printStackTrace();
        }
    }

// recombine and mix each row in a column

    void MixSubColumns (char state[])
```

```
{
 char[] newstate=new char[4 * Nb];

        // mixing column 0
        newstate[0] = (char)(Xtime2Sbox[state[0]%256] ^
Xtime3Sbox[state[5]%256] ^ Sbox[state[10]%256] ^ Sbox[state[15]%256]);
        newstate[1] = (char)(Sbox[state[0]%256] ^ Xtime2Sbox[state[5]%256] ^
Xtime3Sbox[state[10]%256] ^ Sbox[state[15]%256]);
        newstate[2] = (char)(Sbox[state[0]%256] ^ Sbox[state[5]%256] ^
Xtime2Sbox[state[10]%256] ^ Xtime3Sbox[state[15]%256]);
        newstate[3] = (char)(Xtime3Sbox[state[0]%256] ^ Sbox[state[5]%256] ^
Sbox[state[10]%256] ^ Xtime2Sbox[state[15]%256]);

        // mixing column 1
        newstate[4] = (char)(Xtime2Sbox[state[4]%256] ^
Xtime3Sbox[state[9]%256] ^ Sbox[state[14]%256] ^ Sbox[state[3]%256]);
        newstate[5] = (char)(Sbox[state[4]%256] ^ Xtime2Sbox[state[9]%256] ^
Xtime3Sbox[state[14]%256] ^ Sbox[state[3]%256]);
        newstate[6] = (char)(Sbox[state[4]%256] ^ Sbox[state[9]%256] ^
Xtime2Sbox[state[14]%256] ^ Xtime3Sbox[state[3]%256]);
        newstate[7] = (char)(Xtime3Sbox[state[4]%256] ^ Sbox[state[9]%256] ^
Sbox[state[14]%256] ^ Xtime2Sbox[state[3]%256]);

        // mixing column 2
        newstate[8] = (char)(Xtime2Sbox[state[8]%256] ^
Xtime3Sbox[state[13]%256] ^ Sbox[state[2]%256] ^ Sbox[state[7]%256]);
        newstate[9] = (char)(Sbox[state[8]%256] ^ Xtime2Sbox[state[13]%256] ^
Xtime3Sbox[state[2]%256] ^ Sbox[state[7]%256]);
```

```
        newstate[10]  = (char)(Sbox[state[8]%256] ^ Sbox[state[13]%256] ^
Xtime2Sbox[state[2]%256] ^ Xtime3Sbox[state[7]%256]);
        newstate[11]  = (char)(Xtime3Sbox[state[8]%256] ^ Sbox[state[13]%256]
^ Sbox[state[2]%256] ^ Xtime2Sbox[state[7]%256]);

        // mixing column 3
        newstate[12] = (char)(Xtime2Sbox[state[12]%256] ^
Xtime3Sbox[state[1]%256] ^ Sbox[state[6]%256] ^ Sbox[state[11]%256]);
        newstate[13] = (char)(Sbox[state[12]%256] ^ Xtime2Sbox[state[1]%256]
^ Xtime3Sbox[state[6]%256] ^ Sbox[state[11]%256]);
        newstate[14] = (char)(Sbox[state[12]%256] ^ Sbox[state[1]%256] ^
Xtime2Sbox[state[6]%256] ^ Xtime3Sbox[state[11]%256]);
        newstate[15] = (char)(Xtime3Sbox[state[12]%256] ^ Sbox[state[1]%256]
^ Sbox[state[6]%256] ^ Xtime2Sbox[state[11]%256]);



//     memcpy (state, newstate, sizeof(newstate));
        for (int i = 0; i < 4 * Nb; i++) {
     state[i] = newstate[i];
   }
 }

 char Rcon[] = {0x00, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b,
0x36};

// produce Nk bytes for each round
 void ExpandKey ( char key[], char expkey[])
 {
  char tmp0, tmp1, tmp2, tmp3, tmp4;
```

```
int idx;

        for( idx = 0; idx < Nk; idx++ ) {
                expkey[4*idx+0] = key[4 * idx + 0];
                expkey[4*idx+1] = key[4 * idx + 1];
                expkey[4*idx+2] = key[4 * idx + 2];
                expkey[4*idx+3] = key[4 * idx + 3];
        }

        for( idx = Nk; idx < Nb * (Nr + 1); idx++ ) {
                tmp0 = expkey[4*idx - 4];
                tmp1 = expkey[4*idx - 3];
                tmp2 = expkey[4*idx - 2];
                tmp3 = expkey[4*idx - 1];
                if( (idx % Nk)<=0 ) {
                        tmp4 = tmp3;
                        tmp3 = Sbox[tmp0];
                        tmp0 = (char)(Sbox[tmp1] ^ Rcon[idx/Nk]);
                        tmp1 = Sbox[tmp2];
                        tmp2 = Sbox[tmp4];
                }

                //  convert from longs to bytes

                expkey[4*idx+0] =(char)( expkey[4*idx - 4*Nk + 0] ^ tmp0);
                expkey[4*idx+1] = (char)(expkey[4*idx - 4*Nk + 1] ^ tmp1);
                expkey[4*idx+2] = (char)(expkey[4*idx - 4*Nk + 2] ^ tmp2);
                expkey[4*idx+3] =(char)( expkey[4*idx - 4*Nk + 3] ^ tmp3);
        }
```

```
 }

// encrypt/decrypt columns of the key
// n.b. you can replace this with
//      byte-wise xor if you wish.

 void AddRoundKey (char state[], char key[])
 {
 int idx;

        for( idx = 0; idx <16; idx++ )
                state[idx] ^= key[idx];
 }



 static void AddRoundKey(char[] state, char[] expkey, int key) {
   int idx;

   for (idx = 0; idx < 16; idx++) {
     state[idx] ^= expkey[key + idx];
   }

 }


// encrypt one 128 bit block
 byte[] Encrypt (char in[],  char expkey[])
 {
```

```
 char[] out=new char[16];
 int round, idx,i=0;
 char[]  state=new char[Nb * 4];

        for( idx = 0; idx < Nb; idx++ ) {
              state[4*idx+0] = in[i++];
              state[4*idx+1] = in[i++];
              state[4*idx+2] = in[i++];
              state[4*idx+3] = in[i++];
        }

        AddRoundKey (state, expkey);
//          AddRoundKey (state, expkey,160);

        for( round = 1; round < Nr + 1; round++ ) {
              if( round < Nr )
                    MixSubColumns (state);
              else
                    ShiftRows (state);

//                AddRoundKey (state, expkey + round * Nb);
                  AddRoundKey(state, expkey, round * 16);
        }
 int ii=0;
        for( idx = 0; idx < Nb; idx++ ) {
              out[ii++] = state[4*idx+0];
              out[ii++] = state[4*idx+1];
              out[ii++] = state[4*idx+2];
              out[ii++] = state[4*idx+3];
```

```
        }
        return CharToByte(out);
    }


    public static char[] ByteToChar(byte[] data) {
      char[] A = new char[data.length];
      for (int i = 0; i < data.length; i++) {
        A[i] = (char) data[i];
      }
      return A;
    }


     public static byte[] CharToByte(char[] data) {
       byte[] A = new byte[data.length];
       for (int i = 0; i < data.length; i++) {
         A[i] = (byte) data[i];
       }
       return A;
     }



     public byte[] Encrypt_Byte(byte[] in,String key) {
        if(in.length>16||key.length()!=16){
            System.out.println("tidak ditampilkan");
            System.exit(1);
        }
```

```
    char[] KEY=key.toCharArray();
      char[] expkey = new char[4 * Nb * (Nr + 1)];
      byte[] out = new byte[16];
      ExpandKey(KEY, expkey);
      try {
         out = Encrypt(ByteToChar(in), expkey);
      } catch (Exception e) {
         System.out.println("hello");
         e.printStackTrace();
      }
      return out;

  }

  @SuppressWarnings("empty-statement")
            public static void  encryptFile() throws IOException,
ShortBufferException, IllegalBlockSizeException, BadPaddingException
            {

                        int blockSize = cipher.getBlockSize();
               int outputSize = cipher.getOutputSize(blockSize);
               byte[] inBytes = new byte[blockSize];
               byte[] outBytes = new byte[outputSize];
               int progress=0;
               File file = new File(inputFile);
               in= new FileInputStream(inputFile);

               ProgressMonitorInputStream progressIn = new
ProgressMonitorInputStream(dummyFrame,"Proses Enkripsi file...",in);
```

```
                BufferedInputStream inStream = new
BufferedInputStream(progressIn);
                inputFileLength=in.available();
                out=new FileOutputStream(outputFile);
                int inLength = 0;;
                boolean more = true;

                while (more)
                 {
                   inLength = inStream.read(inBytes);
                   if (inLength == blockSize)
                   {
                     int outLength
                        = cipher.update(inBytes, 0, blockSize, outBytes);
                     out.write(outBytes, 0, outLength);
                    // System.out.println(outLength);
                   }
                   else more = false;
                 }
                 if (inLength > 0)
                   outBytes = cipher.doFinal(inBytes, 0, inLength);
                 else
                   outBytes = cipher.doFinal();
            JOptionPane.showMessageDialog(dummyFrame, "Enkripsi Selesai..!!");
                 System.out.println(outBytes.length);
                 out.write(outBytes);

                       if(Key==null||Key.length()>16){
```

```
                return;
    }
    char[] key=Key.toCharArray();
    long startTime = System.nanoTime();
    //char[] key = {'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n','o', 'p'};
    byte[] copy = new byte[16];
    char[] expkey = new char[4 * Nb * (Nr + 1)];
    FileInputStream fp1 = new FileInputStream(Encrypt_Befor);
    FileOutputStream fp2 = new FileOutputStream(Encrypt_After, true);

    long Length = fp1.available();

    long N = 0;
    if (Length != 0) {
    N = 18 - (Length + 2) % 16;
    if (N == 18)
N = 2;
    }
    Length = Length + N;

    long leave = Length / (4 * Nb);
    byte[] state = new byte[16];
    boolean isnull = true;
    boolean indexout = false;
    byte[] bb = new byte[14];

    int first=1;
    while (leave > 0) {
```

```
    if (first == 1) {
       state[0] = (byte) (N / 10);
       state[1] = (byte) (N % 10);
       for (int j = 2; j < N; j++) {
          state[j] = 'A';
       }
       try {
          fp1.read(state, (int) (N), (int) (16 - N));
       } catch (IOException ex) {
          ex.printStackTrace();
       }
       first = 2;
    }
    else {
                        fp1.read(state, 0, 16);
    }

    fp2.write(copy, 0, 16);
    leave--;
    }
    fp1.close();
    fp2.close();
    long timer = (System.nanoTime() - startTime) / 1000000;
    System.out.println("kapasitas panjang:  ....." + Length);
    System.out.println("kapasitas waktu:  ....." + timer);
          }

public static void aksi()
  {
```

```
            //boolean debug = true;
            //DEBUG = debug;
            matchPassword();
            createKey();
            progressMonitorReport();
            progressMonitorprogress();

    }
        public static void openFile(){
                    fd= new FileDialog( dummyFrame, "Pilih File Untuk Di
Enkripsi");
                    fd.setVisible(true);
                    inputFile=fd.getDirectory()+fd.getFile();
        }

            public static void matchPassword()
            {
                    while(true)
                    {
                            pwd1 = new JPasswordField(25);
                            pwd2 = new JPasswordField(25);
                            JOptionPane.showConfirmDialog(null,
pwd1,"Masukkan Password",JOptionPane.OK_CANCEL_OPTION);
                            JOptionPane.showConfirmDialog(null,
pwd2,"Masukkan Password Kembali",JOptionPane.OK_CANCEL_OPTION);
                            outputFile =
JOptionPane.showInputDialog(dummyFrame,"Masukkan Nama untuk File
Enkripsi");
                            passWord1=new String(pwd1.getPassword());
```

```
                    passWord2=new String(pwd2.getPassword());

                if(passWord1.equals(passWord2))
                {
                    manageKeystrengthMethod();
                    passByte=passWord1.getBytes();
                    createExtensionForOutputFile();
                    break;
                }
                else
                {
                    JOptionPane.showMessageDialog(null,"Password
Mismatch");
                    continue;
                }
                }
        }


        public static void  createExtensionForOutputFile()
        {
                inputfileName=fd.getFile();
                int i=0;

                i=inputfileName.indexOf(".");
```

```
        outputFile=outputFile+inputfileName.substring(i,inputfileName.length
());

        }

        public static void createKey()
        {

                try
                {
                        keyFactory =
SecretKeyFactory.getInstance("DES");
                        DESKeySpec dspec= new
DESKeySpec(passByte);
                        key = keyFactory.generateSecret(dspec);
                        //System.out.println("key is :"+key);
                        createCipher();
                        new Thread()
                        {

        @Override
                        public void run()
                        {
                                try
                                {
                        encryptFile();
                                }
                                catch(Exception e)
```

```
                                        {
                                                e.printStackTrace();
                                        }
                                }
                                }.start();
                        }
                        catch(Exception e)
                        {
                                e.printStackTrace();

                        }
}

public  static void  createCipher()
{
                        try
                        {
                        cipher=Cipher.getInstance("DES");
                        cipher.init(Cipher.ENCRYPT_MODE,key);
                        }
                        catch(Exception e)
                        {
                                e.printStackTrace();
                        }

}
public static void manageKeystrengthMethod()
{
                        if(passWord1.length()<8)
```

```
                {
                        int counter=passWord1.length();

                        while(counter<8)
                        {
                                passWord1+='@';
                                counter++;
                        }

                }
        }
   @SuppressWarnings("empty-statement")


        public static void progressMonitorReport()
        {


                                pMonitor= new
ProgressMonitor(dummyFrame,
                        "Encryption in progress",
                        "",
                        0,
                        inputFileLength);


        }

        public  static void progressMonitorprogress()
```

```
		{
			Object[] progress = null;
			String message = String.format("Completed %d%%.\n",
progress);
			pMonitor.setNote(message);

		}
		private static JFrame dummyFrame = new JFrame();
		private static FileDialog fd;
		public static String inputFile;
		private static String passWord1;
		private static String passWord2;
		private  static byte[] passByte;
		private  static SecretKeyFactory keyFactory;
		private  static Key key;
		private static JPasswordField pwd1;
		private static JPasswordField pwd2;
		private static Cipher cipher;
		private static String outputFile;
		private static InputStream in;
		private static OutputStream out;
		private static ProgressMonitor pMonitor;
		private static int inputFileLength;
		private static String inputfileName;

		public static String Encrypt_Befor;
		public static String Encrypt_After;
		public static String Key;
```

```
	private static final String JCE_EXCEPTION_MESSAGE = "Please
make sure "
			+ "\"Java Cryptography Extension (JCE) Unlimited Strength
Jurisdiction Policy Files\" "
			+ "(http://java.sun.com/javase/downloads/index.jsp) is
installed on your JRE.";
	private static final String RANDOM_ALG = "SHA1PRNG";
	private static final String DIGEST_ALG = "SHA-256";
	private static final String HMAC_ALG = "HmacSHA256";
	private static final String CRYPT_ALG = "AES";
	private static final String CRYPT_TRANS = "AES/CBC/NoPadding";
	private static final byte[] DEFAULT_MAC =
			{0x01, 0x23, 0x45, 0x67, (byte) 0x89, (byte) 0xab, (byte)
0xcd, (byte) 0xef};
	private static final int KEY_SIZE = 32;
	private static final int BLOCK_SIZE = 16;
	private static final int SHA_SIZE = 32;

	private byte[] matchPassword;
	private Mac hmac;
	private SecureRandom random;
	private MessageDigest digest;
	private IvParameterSpec ivSpec1;
	private SecretKeySpec aesKey1;
	private IvParameterSpec ivSpec2;
	private SecretKeySpec aesKey2;

}
```

Dekripsi.java

```
import java.awt.FileDialog;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.BufferedInputStream;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.security.Key;

import javax.crypto.BadPaddingException;
import javax.crypto.Cipher;
import javax.crypto.IllegalBlockSizeException;
import javax.crypto.SecretKeyFactory;
import javax.crypto.ShortBufferException;
import javax.crypto.spec.DESKeySpec;
import javax.swing.JFrame;
import javax.swing.JOptionPane;
import javax.swing.JPasswordField;
import javax.swing.ProgressMonitorInputStream;

import java.security.GeneralSecurityException;
import java.security.InvalidKeyException;
import java.io.BufferedOutputStream;
import java.util.Arrays;
```

```
import java.util.Enumeration;
import java.net.NetworkInterface;
import java.io.UnsupportedEncodingException;
import java.security.GeneralSecurityException;
import java.security.SecureRandom;
import java.security.MessageDigest;
import javax.crypto.spec.SecretKeySpec;
import javax.crypto.Mac;
import javax.crypto.spec.IvParameterSpec;
public class Dekripsi {

  static int Nk = 4; // number of columns in a key
  static int Nr = 10; // number of rounds in encryption
  static int Nb = 4;
  static int len = 0;

  static char Sbox[] = { // forward s-box
      0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x01, 0x67, 0x2b,
      0xfe, 0xd7, 0xab, 0x76,
      0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0, 0xad, 0xd4, 0xa2, 0xaf,
      0x9c, 0xa4, 0x72, 0xc0,
      0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc, 0x34, 0xa5, 0xe5, 0xf1,
      0x71, 0xd8, 0x31, 0x15,
      0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a, 0x07, 0x12, 0x80, 0xe2,
      0xeb, 0x27, 0xb2, 0x75,
      0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0, 0x52, 0x3b, 0xd6, 0xb3,
      0x29, 0xe3, 0x2f, 0x84,
      0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b, 0x6a, 0xcb, 0xbe, 0x39,
      0x4a, 0x4c, 0x58, 0xcf,
```

```
    0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85, 0x45, 0xf9, 0x02, 0x7f,
    0x50, 0x3c, 0x9f, 0xa8,
    0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5, 0xbc, 0xb6, 0xda, 0x21,
    0x10, 0xff, 0xf3, 0xd2,
    0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17, 0xc4, 0xa7, 0x7e, 0x3d,
    0x64, 0x5d, 0x19, 0x73,
    0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88, 0x46, 0xee, 0xb8, 0x14,
    0xde, 0x5e, 0x0b, 0xdb,
    0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c, 0xc2, 0xd3, 0xac, 0x62,
    0x91, 0x95, 0xe4, 0x79,
    0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9, 0x6c, 0x56, 0xf4, 0xea,
    0x65, 0x7a, 0xae, 0x08,
    0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6, 0xe8, 0xdd, 0x74, 0x1f,
    0x4b, 0xbd, 0x8b, 0x8a,
    0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e, 0x61, 0x35, 0x57, 0xb9,
    0x86, 0xc1, 0x1d, 0x9e,
    0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94, 0x9b, 0x1e, 0x87, 0xe9,
    0xce, 0x55, 0x28, 0xdf,
    0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68, 0x41, 0x99, 0x2d, 0x0f,
    0xb0, 0x54, 0xbb, 0x16};

static char InvSbox[] = { // inverse s-box
    0x52, 0x09, 0x6a, 0xd5, 0x30, 0x36, 0xa5, 0x38, 0xbf, 0x40, 0xa3, 0x9e,
    0x81, 0xf3, 0xd7, 0xfb,
    0x7c, 0xe3, 0x39, 0x82, 0x9b, 0x2f, 0xff, 0x87, 0x34, 0x8e, 0x43, 0x44,
    0xc4, 0xde, 0xe9, 0xcb,
    0x54, 0x7b, 0x94, 0x32, 0xa6, 0xc2, 0x23, 0x3d, 0xee, 0x4c, 0x95, 0x0b,
    0x42, 0xfa, 0xc3, 0x4e,
    0x08, 0x2e, 0xa1, 0x66, 0x28, 0xd9, 0x24, 0xb2, 0x76, 0x5b, 0xa2, 0x49,
```

```
    0x6d, 0x8b, 0xd1, 0x25,
    0x72, 0xf8, 0xf6, 0x64, 0x86, 0x68, 0x98, 0x16, 0xd4, 0xa4, 0x5c, 0xcc,
    0x5d, 0x65, 0xb6, 0x92,
    0x6c, 0x70, 0x48, 0x50, 0xfd, 0xed, 0xb9, 0xda, 0x5e, 0x15, 0x46, 0x57,
    0xa7, 0x8d, 0x9d, 0x84,
    0x90, 0xd8, 0xab, 0x00, 0x8c, 0xbc, 0xd3, 0x0a, 0xf7, 0xe4, 0x58, 0x05,
    0xb8, 0xb3, 0x45, 0x06,
    0xd0, 0x2c, 0x1e, 0x8f, 0xca, 0x3f, 0x0f, 0x02, 0xc1, 0xaf, 0xbd, 0x03,
    0x01, 0x13, 0x8a, 0x6b,
    0x3a, 0x91, 0x11, 0x41, 0x4f, 0x67, 0xdc, 0xea, 0x97, 0xf2, 0xcf, 0xce,
    0xf0, 0xb4, 0xe6, 0x73,
    0x96, 0xac, 0x74, 0x22, 0xe7, 0xad, 0x35, 0x85, 0xe2, 0xf9, 0x37, 0xe8,
    0x1c, 0x75, 0xdf, 0x6e,
    0x47, 0xf1, 0x1a, 0x71, 0x1d, 0x29, 0xc5, 0x89, 0x6f, 0xb7, 0x62, 0x0e,
    0xaa, 0x18, 0xbe, 0x1b,
    0xfc, 0x56, 0x3e, 0x4b, 0xc6, 0xd2, 0x79, 0x20, 0x9a, 0xdb, 0xc0, 0xfe,
    0x78, 0xcd, 0x5a, 0xf4,
    0x1f, 0xdd, 0xa8, 0x33, 0x88, 0x07, 0xc7, 0x31, 0xb1, 0x12, 0x10, 0x59,
    0x27, 0x80, 0xec, 0x5f,
    0x60, 0x51, 0x7f, 0xa9, 0x19, 0xb5, 0x4a, 0x0d, 0x2d, 0xe5, 0x7a, 0x9f,
    0x93, 0xc9, 0x9c, 0xef,
    0xa0, 0xe0, 0x3b, 0x4d, 0xae, 0x2a, 0xf5, 0xb0, 0xc8, 0xeb, 0xbb, 0x3c,
    0x83, 0x53, 0x99, 0x61,
    0x17, 0x2b, 0x04, 0x7e, 0xba, 0x77, 0xd6, 0x26, 0xe1, 0x69, 0x14, 0x63,
    0x55, 0x21, 0x0c, 0x7d};

static char Xtime2[] = {
    0x00, 0x02, 0x04, 0x06, 0x08, 0x0a, 0x0c, 0x0e, 0x10, 0x12, 0x14, 0x16,
    0x18, 0x1a, 0x1c, 0x1e,
```

0x20, 0x22, 0x24, 0x26, 0x28, 0x2a, 0x2c, 0x2e, 0x30, 0x32, 0x34, 0x36,
0x38, 0x3a, 0x3c, 0x3e,
0x40, 0x42, 0x44, 0x46, 0x48, 0x4a, 0x4c, 0x4e, 0x50, 0x52, 0x54, 0x56,
0x58, 0x5a, 0x5c, 0x5e,
0x60, 0x62, 0x64, 0x66, 0x68, 0x6a, 0x6c, 0x6e, 0x70, 0x72, 0x74, 0x76,
0x78, 0x7a, 0x7c, 0x7e,
0x80, 0x82, 0x84, 0x86, 0x88, 0x8a, 0x8c, 0x8e, 0x90, 0x92, 0x94, 0x96,
0x98, 0x9a, 0x9c, 0x9e,
0xa0, 0xa2, 0xa4, 0xa6, 0xa8, 0xaa, 0xac, 0xae, 0xb0, 0xb2, 0xb4, 0xb6,
0xb8, 0xba, 0xbc, 0xbe,
0xc0, 0xc2, 0xc4, 0xc6, 0xc8, 0xca, 0xcc, 0xce, 0xd0, 0xd2, 0xd4, 0xd6,
0xd8, 0xda, 0xdc, 0xde,
0xe0, 0xe2, 0xe4, 0xe6, 0xe8, 0xea, 0xec, 0xee, 0xf0, 0xf2, 0xf4, 0xf6,
0xf8, 0xfa, 0xfc, 0xfe,
0x1b, 0x19, 0x1f, 0x1d, 0x13, 0x11, 0x17, 0x15, 0x0b, 0x09, 0x0f, 0x0d,
0x03, 0x01, 0x07, 0x05,
0x3b, 0x39, 0x3f, 0x3d, 0x33, 0x31, 0x37, 0x35, 0x2b, 0x29, 0x2f, 0x2d,
0x23, 0x21, 0x27, 0x25,
0x5b, 0x59, 0x5f, 0x5d, 0x53, 0x51, 0x57, 0x55, 0x4b, 0x49, 0x4f, 0x4d,
0x43, 0x41, 0x47, 0x45,
0x7b, 0x79, 0x7f, 0x7d, 0x73, 0x71, 0x77, 0x75, 0x6b, 0x69, 0x6f, 0x6d,
0x63, 0x61, 0x67, 0x65,
0x9b, 0x99, 0x9f, 0x9d, 0x93, 0x91, 0x97, 0x95, 0x8b, 0x89, 0x8f, 0x8d,
0x83, 0x81, 0x87, 0x85,
0xbb, 0xb9, 0xbf, 0xbd, 0xb3, 0xb1, 0xb7, 0xb5, 0xab, 0xa9, 0xaf, 0xad,
0xa3, 0xa1, 0xa7, 0xa5,
0xdb, 0xd9, 0xdf, 0xdd, 0xd3, 0xd1, 0xd7, 0xd5, 0xcb, 0xc9, 0xcf, 0xcd,
0xc3, 0xc1, 0xc7, 0xc5,
0xfb, 0xf9, 0xff, 0xfd, 0xf3, 0xf1, 0xf7, 0xf5, 0xeb, 0xe9, 0xef, 0xed,

```
    0xe3, 0xe1, 0xe7, 0xe5};

static char Xtime9[] = {
    0x00, 0x09, 0x12, 0x1b, 0x24, 0x2d, 0x36, 0x3f, 0x48, 0x41, 0x5a, 0x53,
    0x6c, 0x65, 0x7e, 0x77,
    0x90, 0x99, 0x82, 0x8b, 0xb4, 0xbd, 0xa6, 0xaf, 0xd8, 0xd1, 0xca, 0xc3,
    0xfc, 0xf5, 0xee, 0xe7,
    0x3b, 0x32, 0x29, 0x20, 0x1f, 0x16, 0x0d, 0x04, 0x73, 0x7a, 0x61, 0x68,
    0x57, 0x5e, 0x45, 0x4c,
    0xab, 0xa2, 0xb9, 0xb0, 0x8f, 0x86, 0x9d, 0x94, 0xe3, 0xea, 0xf1, 0xf8,
    0xc7, 0xce, 0xd5, 0xdc,
    0x76, 0x7f, 0x64, 0x6d, 0x52, 0x5b, 0x40, 0x49, 0x3e, 0x37, 0x2c, 0x25,
    0x1a, 0x13, 0x08, 0x01,
    0xe6, 0xef, 0xf4, 0xfd, 0xc2, 0xcb, 0xd0, 0xd9, 0xae, 0xa7, 0xbc, 0xb5,
    0x8a, 0x83, 0x98, 0x91,
    0x4d, 0x44, 0x5f, 0x56, 0x69, 0x60, 0x7b, 0x72, 0x05, 0x0c, 0x17, 0x1e,
    0x21, 0x28, 0x33, 0x3a,
    0xdd, 0xd4, 0xcf, 0xc6, 0xf9, 0xf0, 0xeb, 0xe2, 0x95, 0x9c, 0x87, 0x8e,
    0xb1, 0xb8, 0xa3, 0xaa,
    0xec, 0xe5, 0xfe, 0xf7, 0xc8, 0xc1, 0xda, 0xd3, 0xa4, 0xad, 0xb6, 0xbf,
    0x80, 0x89, 0x92, 0x9b,
    0x7c, 0x75, 0x6e, 0x67, 0x58, 0x51, 0x4a, 0x43, 0x34, 0x3d, 0x26, 0x2f,
    0x10, 0x19, 0x02, 0x0b,
    0xd7, 0xde, 0xc5, 0xcc, 0xf3, 0xfa, 0xe1, 0xe8, 0x9f, 0x96, 0x8d, 0x84,
    0xbb, 0xb2, 0xa9, 0xa0,
    0x47, 0x4e, 0x55, 0x5c, 0x63, 0x6a, 0x71, 0x78, 0x0f, 0x06, 0x1d, 0x14,
    0x2b, 0x22, 0x39, 0x30,
    0x9a, 0x93, 0x88, 0x81, 0xbe, 0xb7, 0xac, 0xa5, 0xd2, 0xdb, 0xc0, 0xc9,
    0xf6, 0xff, 0xe4, 0xed,
```

```
    0x0a, 0x03, 0x18, 0x11, 0x2e, 0x27, 0x3c, 0x35, 0x42, 0x4b, 0x50, 0x59,
    0x66, 0x6f, 0x74, 0x7d,
    0xa1, 0xa8, 0xb3, 0xba, 0x85, 0x8c, 0x97, 0x9e, 0xe9, 0xe0, 0xfb, 0xf2,
    0xcd, 0xc4, 0xdf, 0xd6,
    0x31, 0x38, 0x23, 0x2a, 0x15, 0x1c, 0x07, 0x0e, 0x79, 0x70, 0x6b, 0x62,
    0x5d, 0x54, 0x4f, 0x46};

static char XtimeB[] = {
    0x00, 0x0b, 0x16, 0x1d, 0x2c, 0x27, 0x3a, 0x31, 0x58, 0x53, 0x4e, 0x45,
    0x74, 0x7f, 0x62, 0x69,
    0xb0, 0xbb, 0xa6, 0xad, 0x9c, 0x97, 0x8a, 0x81, 0xe8, 0xe3, 0xfe, 0xf5,
    0xc4, 0xcf, 0xd2, 0xd9,
    0x7b, 0x70, 0x6d, 0x66, 0x57, 0x5c, 0x41, 0x4a, 0x23, 0x28, 0x35, 0x3e,
    0x0f, 0x04, 0x19, 0x12,
    0xcb, 0xc0, 0xdd, 0xd6, 0xe7, 0xec, 0xf1, 0xfa, 0x93, 0x98, 0x85, 0x8e,
    0xbf, 0xb4, 0xa9, 0xa2,
    0xf6, 0xfd, 0xe0, 0xeb, 0xda, 0xd1, 0xcc, 0xc7, 0xae, 0xa5, 0xb8, 0xb3,
    0x82, 0x89, 0x94, 0x9f,
    0x46, 0x4d, 0x50, 0x5b, 0x6a, 0x61, 0x7c, 0x77, 0x1e, 0x15, 0x08, 0x03,
    0x32, 0x39, 0x24, 0x2f,
    0x8d, 0x86, 0x9b, 0x90, 0xa1, 0xaa, 0xb7, 0xbc, 0xd5, 0xde, 0xc3, 0xc8,
    0xf9, 0xf2, 0xef, 0xe4,
    0x3d, 0x36, 0x2b, 0x20, 0x11, 0x1a, 0x07, 0x0c, 0x65, 0x6e, 0x73, 0x78,
    0x49, 0x42, 0x5f, 0x54,
    0xf7, 0xfc, 0xe1, 0xea, 0xdb, 0xd0, 0xcd, 0xc6, 0xaf, 0xa4, 0xb9, 0xb2,
    0x83, 0x88, 0x95, 0x9e,
    0x47, 0x4c, 0x51, 0x5a, 0x6b, 0x60, 0x7d, 0x76, 0x1f, 0x14, 0x09, 0x02,
    0x33, 0x38, 0x25, 0x2e,
    0x8c, 0x87, 0x9a, 0x91, 0xa0, 0xab, 0xb6, 0xbd, 0xd4, 0xdf, 0xc2, 0xc9,
```

```
    0xf8, 0xf3, 0xee, 0xe5,
    0x3c, 0x37, 0x2a, 0x21, 0x10, 0x1b, 0x06, 0x0d, 0x64, 0x6f, 0x72, 0x79,
    0x48, 0x43, 0x5e, 0x55,
    0x01, 0x0a, 0x17, 0x1c, 0x2d, 0x26, 0x3b, 0x30, 0x59, 0x52, 0x4f, 0x44,
    0x75, 0x7e, 0x63, 0x68,
    0xb1, 0xba, 0xa7, 0xac, 0x9d, 0x96, 0x8b, 0x80, 0xe9, 0xe2, 0xff, 0xf4,
    0xc5, 0xce, 0xd3, 0xd8,
    0x7a, 0x71, 0x6c, 0x67, 0x56, 0x5d, 0x40, 0x4b, 0x22, 0x29, 0x34, 0x3f,
    0x0e, 0x05, 0x18, 0x13,
    0xca, 0xc1, 0xdc, 0xd7, 0xe6, 0xed, 0xf0, 0xfb, 0x92, 0x99, 0x84, 0x8f,
    0xbe, 0xb5, 0xa8, 0xa3};

static char XtimeD[] = {
    0x00, 0x0d, 0x1a, 0x17, 0x34, 0x39, 0x2e, 0x23, 0x68, 0x65, 0x72, 0x7f,
    0x5c, 0x51, 0x46, 0x4b,
    0xd0, 0xdd, 0xca, 0xc7, 0xe4, 0xe9, 0xfe, 0xf3, 0xb8, 0xb5, 0xa2, 0xaf,
    0x8c, 0x81, 0x96, 0x9b,
    0xbb, 0xb6, 0xa1, 0xac, 0x8f, 0x82, 0x95, 0x98, 0xd3, 0xde, 0xc9, 0xc4,
    0xe7, 0xea, 0xfd, 0xf0,
    0x6b, 0x66, 0x71, 0x7c, 0x5f, 0x52, 0x45, 0x48, 0x03, 0x0e, 0x19, 0x14,
    0x37, 0x3a, 0x2d, 0x20,
    0x6d, 0x60, 0x77, 0x7a, 0x59, 0x54, 0x43, 0x4e, 0x05, 0x08, 0x1f, 0x12,
    0x31, 0x3c, 0x2b, 0x26,
    0xbd, 0xb0, 0xa7, 0xaa, 0x89, 0x84, 0x93, 0x9e, 0xd5, 0xd8, 0xcf, 0xc2,
    0xe1, 0xec, 0xfb, 0xf6,
    0xd6, 0xdb, 0xcc, 0xc1, 0xe2, 0xef, 0xf8, 0xf5, 0xbe, 0xb3, 0xa4, 0xa9,
    0x8a, 0x87, 0x90, 0x9d,
    0x06, 0x0b, 0x1c, 0x11, 0x32, 0x3f, 0x28, 0x25, 0x6e, 0x63, 0x74, 0x79,
    0x5a, 0x57, 0x40, 0x4d,
```

```
    0xda, 0xd7, 0xc0, 0xcd, 0xee, 0xe3, 0xf4, 0xf9, 0xb2, 0xbf, 0xa8, 0xa5,
    0x86, 0x8b, 0x9c, 0x91,
    0x0a, 0x07, 0x10, 0x1d, 0x3e, 0x33, 0x24, 0x29, 0x62, 0x6f, 0x78, 0x75,
    0x56, 0x5b, 0x4c, 0x41,
    0x61, 0x6c, 0x7b, 0x76, 0x55, 0x58, 0x4f, 0x42, 0x09, 0x04, 0x13, 0x1e,
    0x3d, 0x30, 0x27, 0x2a,
    0xb1, 0xbc, 0xab, 0xa6, 0x85, 0x88, 0x9f, 0x92, 0xd9, 0xd4, 0xc3, 0xce,
    0xed, 0xe0, 0xf7, 0xfa,
    0xb7, 0xba, 0xad, 0xa0, 0x83, 0x8e, 0x99, 0x94, 0xdf, 0xd2, 0xc5, 0xc8,
    0xeb, 0xe6, 0xf1, 0xfc,
    0x67, 0x6a, 0x7d, 0x70, 0x53, 0x5e, 0x49, 0x44, 0x0f, 0x02, 0x15, 0x18,
    0x3b, 0x36, 0x21, 0x2c,
    0x0c, 0x01, 0x16, 0x1b, 0x38, 0x35, 0x22, 0x2f, 0x64, 0x69, 0x7e, 0x73,
    0x50, 0x5d, 0x4a, 0x47,
    0xdc, 0xd1, 0xc6, 0xcb, 0xe8, 0xe5, 0xf2, 0xff, 0xb4, 0xb9, 0xae, 0xa3,
    0x80, 0x8d, 0x9a, 0x97};

static char XtimeE[] = {
    0x00, 0x0e, 0x1c, 0x12, 0x38, 0x36, 0x24, 0x2a, 0x70, 0x7e, 0x6c, 0x62,
    0x48, 0x46, 0x54, 0x5a,
    0xe0, 0xee, 0xfc, 0xf2, 0xd8, 0xd6, 0xc4, 0xca, 0x90, 0x9e, 0x8c, 0x82,
    0xa8, 0xa6, 0xb4, 0xba,
    0xdb, 0xd5, 0xc7, 0xc9, 0xe3, 0xed, 0xff, 0xf1, 0xab, 0xa5, 0xb7, 0xb9,
    0x93, 0x9d, 0x8f, 0x81,
    0x3b, 0x35, 0x27, 0x29, 0x03, 0x0d, 0x1f, 0x11, 0x4b, 0x45, 0x57, 0x59,
    0x73, 0x7d, 0x6f, 0x61,
    0xad, 0xa3, 0xb1, 0xbf, 0x95, 0x9b, 0x89, 0x87, 0xdd, 0xd3, 0xc1, 0xcf,
    0xe5, 0xeb, 0xf9, 0xf7,
    0x4d, 0x43, 0x51, 0x5f, 0x75, 0x7b, 0x69, 0x67, 0x3d, 0x33, 0x21, 0x2f,
```

```
    0x05, 0x0b, 0x19, 0x17,
    0x76, 0x78, 0x6a, 0x64, 0x4e, 0x40, 0x52, 0x5c, 0x06, 0x08, 0x1a, 0x14,
    0x3e, 0x30, 0x22, 0x2c,
    0x96, 0x98, 0x8a, 0x84, 0xae, 0xa0, 0xb2, 0xbc, 0xe6, 0xe8, 0xfa, 0xf4,
    0xde, 0xd0, 0xc2, 0xcc,
    0x41, 0x4f, 0x5d, 0x53, 0x79, 0x77, 0x65, 0x6b, 0x31, 0x3f, 0x2d, 0x23,
    0x09, 0x07, 0x15, 0x1b,
    0xa1, 0xaf, 0xbd, 0xb3, 0x99, 0x97, 0x85, 0x8b, 0xd1, 0xdf, 0xcd, 0xc3,
    0xe9, 0xe7, 0xf5, 0xfb,
    0x9a, 0x94, 0x86, 0x88, 0xa2, 0xac, 0xbe, 0xb0, 0xea, 0xe4, 0xf6, 0xf8,
    0xd2, 0xdc, 0xce, 0xc0,
    0x7a, 0x74, 0x66, 0x68, 0x42, 0x4c, 0x5e, 0x50, 0x0a, 0x04, 0x16, 0x18,
    0x32, 0x3c, 0x2e, 0x20,
    0xec, 0xe2, 0xf0, 0xfe, 0xd4, 0xda, 0xc8, 0xc6, 0x9c, 0x92, 0x80, 0x8e,
    0xa4, 0xaa, 0xb8, 0xb6,
    0x0c, 0x02, 0x10, 0x1e, 0x34, 0x3a, 0x28, 0x26, 0x7c, 0x72, 0x60, 0x6e,
    0x44, 0x4a, 0x58, 0x56,
    0x37, 0x39, 0x2b, 0x25, 0x0f, 0x01, 0x13, 0x1d, 0x47, 0x49, 0x5b, 0x55,
    0x7f, 0x71, 0x63, 0x6d,
    0xd7, 0xd9, 0xcb, 0xc5, 0xef, 0xe1, 0xf3, 0xfd, 0xa7, 0xa9, 0xbb, 0xb5,
    0x9f, 0x91, 0x83, 0x8d};

static void InvShiftRows(char[] state) {
  char tmp;

  state[0] = InvSbox[state[0] % 256];
  state[4] = InvSbox[state[4] % 256];
  state[8] = InvSbox[state[8] % 256];
  state[12] = InvSbox[state[12] % 256];
```

```
    tmp = InvSbox[state[13] % 256];
    state[13] = InvSbox[state[9] % 256];
    state[9] = InvSbox[state[5] % 256];
    state[5] = InvSbox[state[1] % 256];
    state[1] = tmp;

    tmp = InvSbox[state[2] % 256];
    state[2] = InvSbox[state[10] % 256];
    state[10] = tmp;
    tmp = InvSbox[state[6] % 256];
    state[6] = InvSbox[state[14] % 256];
    state[14] = tmp;

    tmp = InvSbox[state[3] % 256];
    state[3] = InvSbox[state[7] % 256];
    state[7] = InvSbox[state[11] % 256];
    state[11] = InvSbox[state[15] % 256];
    state[15] = tmp;

  }

            // recombine and mix each row in a column

// restore and un-mix each row in a column
  static void InvMixSubColumns(char[] state) {
    char[] newstate = new char[16];
    int i;
```

```
   // restore column 0
   newstate[0] = (char) (XtimeE[state[0]] ^ XtimeB[state[1]] ^ XtimeD[state[2]]
^
                        Xtime9[state[3]]);
   newstate[5] = (char) (Xtime9[state[0]] ^ XtimeE[state[1]] ^ XtimeB[state[2]]
^
                        XtimeD[state[3]]);
   newstate[10] = (char) (XtimeD[state[0]] ^ Xtime9[state[1]] ^
XtimeE[state[2]] ^
                         XtimeB[state[3]]);
   newstate[15] = (char) (XtimeB[state[0]] ^ XtimeD[state[1]] ^
Xtime9[state[2]] ^
                         XtimeE[state[3]]);

   // restore column 1
   newstate[4] = (char) (XtimeE[state[4]] ^ XtimeB[state[5]] ^ XtimeD[state[6]]
^
                        Xtime9[state[7]]);
   newstate[9] = (char) (Xtime9[state[4]] ^ XtimeE[state[5]] ^ XtimeB[state[6]]
^
                        XtimeD[state[7]]);
   newstate[14] = (char) (XtimeD[state[4]] ^ Xtime9[state[5]] ^
XtimeE[state[6]] ^
                         XtimeB[state[7]]);
   newstate[3] = (char) (XtimeB[state[4]] ^ XtimeD[state[5]] ^ Xtime9[state[6]]
^
                        XtimeE[state[7]]);

   // restore column 2
```

```
    newstate[8] = (char) (XtimeE[state[8]] ^ XtimeB[state[9]] ^
XtimeD[state[10]] ^
                        Xtime9[state[11]]);
    newstate[13] = (char) (Xtime9[state[8]] ^ XtimeE[state[9]] ^
                         XtimeB[state[10]] ^ XtimeD[state[11]]);
    newstate[2] = (char) (XtimeD[state[8]] ^ Xtime9[state[9]] ^
XtimeE[state[10]] ^
                        XtimeB[state[11]]);
    newstate[7] = (char) (XtimeB[state[8]] ^ XtimeD[state[9]] ^
Xtime9[state[10]] ^
                        XtimeE[state[11]]);

    // restore column 3
    newstate[12] = (char) (XtimeE[state[12]] ^ XtimeB[state[13]] ^
                         XtimeD[state[14]] ^ Xtime9[state[15]]);
    newstate[1] = (char) (Xtime9[state[12]] ^ XtimeE[state[13]] ^
                        XtimeB[state[14]] ^ XtimeD[state[15]]);
    newstate[6] = (char) (XtimeD[state[12]] ^ Xtime9[state[13]] ^
                        XtimeE[state[14]] ^ XtimeB[state[15]]);
    newstate[11] = (char) (XtimeB[state[12]] ^ XtimeD[state[13]] ^
                         Xtime9[state[14]] ^ XtimeE[state[15]]);

    for (i = 0; i < 4 * Nb; i++) {
      state[i] = InvSbox[newstate[i]];
    }
  }
            // encrypt/decrypt columns of the key
// n.b. you can replace this with
//      byte-wise xor if you wish.
```

```
  static char Rcon[] = {
     0x00, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36};

// produce Nk bytes for each round
  public static void ExpandKey(char[] key, char[] expkey) {
    char tmp0, tmp1, tmp2, tmp3, tmp4;
    int idx;

    for (idx = 0; idx < Nk; idx++) {
      expkey[4 * idx + 0] = key[4 * idx + 0];
      expkey[4 * idx + 1] = key[4 * idx + 1];
      expkey[4 * idx + 2] = key[4 * idx + 2];
      expkey[4 * idx + 3] = key[4 * idx + 3];
    }

    for (idx = Nk; idx < Nb * (Nr + 1); idx++) {
      tmp0 = expkey[4 * idx - 4];
      tmp1 = expkey[4 * idx - 3];
      tmp2 = expkey[4 * idx - 2];
      tmp3 = expkey[4 * idx - 1];

      if ( (idx % Nk) <= 0) {
        tmp4 = tmp3;
        tmp3 = Sbox[tmp0];
        tmp0 = (char) (Sbox[tmp1] ^ Rcon[idx / Nk]);
        tmp1 = Sbox[tmp2];
```

```
    tmp2 = Sbox[tmp4];
   }

   //  convert from longs to bytes

   expkey[4 * idx + 0] = (char) (expkey[4 * idx - 4 * Nk + 0] ^ tmp0);
   expkey[4 * idx + 1] = (char) (expkey[4 * idx - 4 * Nk + 1] ^ tmp1);
   expkey[4 * idx + 2] = (char) (expkey[4 * idx - 4 * Nk + 2] ^ tmp2);
   expkey[4 * idx + 3] = (char) (expkey[4 * idx - 4 * Nk + 3] ^ tmp3);
  }
}

        static void AddRoundKey(char[] state, char[] expkey, int key) {
  int idx;

  for (idx = 0; idx < 16; idx++) {
   state[idx] ^= expkey[key + idx];
  }

}

        public static char[] ByteToChar(byte[] data) {
  char[] A = new char[data.length];
  for (int i = 0; i < data.length; i++) {
   A[i] = (char) data[i];
  }
  return A;
}
```

```
public static byte[] CharToByte(char[] data) {
  byte[] A = new byte[data.length];
  for (int i = 0; i < data.length; i++) {
    A[i] = (byte) data[i];
  }
  return A;
}

        public static byte[] Decrypt(char[] in, char[] expkey) {
 char[] out = new char[16];
 int idx, round, j;
 char state[] = new char[Nb * 4];
 int i = 0;
 for (idx = 0; idx < Nb; idx++) {
  state[4 * idx + 0] = in[i++];
  state[4 * idx + 1] = in[i++];
  state[4 * idx + 2] = in[i++];
  state[4 * idx + 3] = in[i++];

 }
 // byte[] a=CharToByte(state);
 //System.out.println((int)a[0]);
 try {
  AddRoundKey(state, expkey, 160); //////////////////////
  byte[] b = CharToByte(state);
  //  System.out.println((int)b[0]+"----------AddRoundKey");
 }
 catch (Exception e) {
```

```
  System.out.println("fuck");
 }
 round = Nr;
 try {
  InvShiftRows(state);
  byte[] c = CharToByte(state);
  //    System.out.println((int)c[0]+"----------InvShiftRows");
 }
 catch (Exception e) {
  System.out.println("fuck22222");
 }
 while (round-- > 0) {
  AddRoundKey(state, expkey, round * 16); ////////////////////////////
  if (round > 0) {
    InvMixSubColumns(state);
  }
 }
 // System.out.println((int)state[0]+"----------while");
 int n = 0;
 for (idx = 0; idx < Nb; idx++) {
  out[n++] = state[4 * idx + 0];
  out[n++] = state[4 * idx + 1];
  out[n++] = state[4 * idx + 2];
  out[n++] = state[4 * idx + 3];
 }

   return CharToByte(out);

}
```

```
 private Cipher aes;
        //private SecretKey aeskey;
        public static void aksi()
 {
        matchPassword();
        createKey();

 }

 static void openFile() {

        fd= new FileDialog( dummyFrame, "Pilih File Untuk Di Dekripsi");
        fd.setVisible(true);
        inputFile=fd.getDirectory()+fd.getFile();


 }

        private static void matchPassword()
        {
                while(true)
                {
                        pwd1 = new JPasswordField(25);
                        pwd2 = new JPasswordField(25);
                        JOptionPane.showConfirmDialog(null,
pwd1,"Enter Password",JOptionPane.OK_CANCEL_OPTION);
                        JOptionPane.showConfirmDialog(null, pwd2,"
Enter Password Again",JOptionPane.OK_CANCEL_OPTION);
```

```
                    outputFile =
JOptionPane.showInputDialog(dummyFrame,"Enter name of output decrypted
file");
                    passWord1=new String(pwd1.getPassword());


                    passWord2=new String(pwd2.getPassword());

            if(passWord1.equals(passWord2))
            {
                    manageKeystrengthMethod();
                    passByte=passWord1.getBytes();
                    createExtensionForOutputFile();
                    break;
            }
            else
            {
                    JOptionPane.showMessageDialog(null,"Password
Salah Tolong Ulangi");
                    continue;
            }
            }
    }

    private static void  createExtensionForOutputFile()
    {
            inputfileName=fd.getFile();
            int i=0;
```

```
                    i=inputfileName.indexOf(".");


          outputFile=outputFile+inputfileName.substring(i,inputfileName.length
());

          }


          private static void createKey()
          {

                    try
                    {

                              keyFactory =
SecretKeyFactory.getInstance("DES");
                              DESKeySpec dspec= new
DESKeySpec(passByte);
                              key = keyFactory.generateSecret(dspec);
                              //System.out.println("key is :"+key);
                              createCipher();

                              new Thread()
                              {

                              public void run()
                              {
```

```
                    try
                    {
                    decryptFile();
                    }
                    catch(Exception e)
                    {
                            e.printStackTrace();
                    }
                    }
                    }.start();
            }
            catch(Exception e)
            {
                    e.printStackTrace();

            }
}
private static void manageKeystrengthMethod()
{
            if(passWord1.length()<8)
            {
                    int counter=passWord1.length();

                    while(counter<8)
                    {
                            passWord1+='@';
                            counter++;
                    }
```

```
                }
        }
        private  static void  createCipher()
        {
                try
                {
                cipher=Cipher.getInstance("DES");
                cipher.init(Cipher.DECRYPT_MODE,key);
                }
                catch(Exception e)
                {
                        e.printStackTrace();
                }

        }
        private static void  decryptFile() throws IOException,
ShortBufferException, IllegalBlockSizeException, BadPaddingException
        {

                int blockSize = cipher.getBlockSize();
           int outputSize = cipher.getOutputSize(blockSize);
           byte[] inBytes = new byte[blockSize];
           byte[] outBytes = new byte[outputSize];
           in= new FileInputStream(inputFile);
           out=new FileOutputStream(outputFile);
           ProgressMonitorInputStream progressIn = new
ProgressMonitorInputStream(dummyFrame,"Proses Dekripsi file...",in);
```

```
                BufferedInputStream inStream = new
BufferedInputStream(progressIn);
                int inLength = 0;;
                boolean more = true;
                while (more)
                 {
                   inLength = inStream.read(inBytes);
                   if (inLength == blockSize)
                   {
                     int outLength
                        = cipher.update(inBytes, 0, blockSize, outBytes);
                     out.write(outBytes, 0, outLength);

                   }
                   else more = false;
                 }
                 if (inLength > 0)
                   outBytes = cipher.doFinal(inBytes, 0, inLength);
                 else
                   outBytes = cipher.doFinal();
          JOptionPane.showMessageDialog(dummyFrame, "Dekripsi
Selsesai..!!");

                        out.write(outBytes);

    if(Key==null||Key.length()>16){
    return;
    }
   char[] key=Key.toCharArray();
```

```
long startTime = System.nanoTime();
//char[] key = {'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n','o', 'p'};
byte[] copy = new byte[16];
char[] expkey = new char[4 * Nb * (Nr + 1)];
FileInputStream fp1 = new FileInputStream(Encrypt_Befor);
FileOutputStream fp2 = new FileOutputStream(Encrypt_After, true);

long Length = fp1.available();

long N = 0;
if (Length != 0) {
   N = 18 - (Length + 2) % 16;
   if (N == 18)
      N = 2;
}
Length = Length + N;

long leave = Length / (4 * Nb);
byte[] state = new byte[16];
boolean isnull = true;
boolean indexout = false;
byte[] bb = new byte[14];

int first=1;
while (leave > 0) {

   if (first == 1) {
      state[0] = (byte) (N / 10);
      state[1] = (byte) (N % 10);
```

```
        for (int j = 2; j < N; j++) {
            state[j] = 'A';
        }
        try {
            fp1.read(state, (int) (N), (int) (16 - N));
        } catch (IOException ex) {
            ex.printStackTrace();
        }
        first = 2;
    } else {
   fp1.read(state, 0, 16);
   }

  fp2.write(copy, 0, 16);
  leave--;
             }
fp1.close();
fp2.close();
long timer = (System.nanoTime() - startTime) / 1000000;
System.out.println("kapasitas:  ....." + Length);
System.out.println("kapasitas:  ....." + timer);

            }
            private static JFrame dummyFrame = new JFrame();
            private static FileDialog fd;
            public static String inputFile;
            private static String passWord1;
            private static String passWord2;
            private static byte[] passByte;
```

```
	private static SecretKeyFactory keyFactory;
	private static Key key;
	private static JPasswordField pwd1;
	private static JPasswordField pwd2;
	private static Cipher cipher;
	private static String outputFile;
	private static InputStream in;
	private static OutputStream out;
	private static String inputfileName;

	public static String Encrypt_Befor;
	public static String Encrypt_After;
	public static String Key;

	private static final String JCE_EXCEPTION_MESSAGE = "Please
make sure "
			+ "\"Java Cryptography Extension (JCE) Unlimited Strength
Jurisdiction Policy Files\" "
			+ "(http://java.sun.com/javase/downloads/index.jsp) is
installed on your JRE.";
	private static final String RANDOM_ALG = "SHA1PRNG";
	private static final String DIGEST_ALG = "SHA-256";
	private static final String HMAC_ALG = "HmacSHA256";
	private static final String CRYPT_ALG = "AES";
	private static final String CRYPT_TRANS = "AES/CBC/NoPadding";
	private static final byte[] DEFAULT_MAC =
			{0x01, 0x23, 0x45, 0x67, (byte) 0x89, (byte) 0xab, (byte)
0xcd, (byte) 0xef};
	private static final int KEY_SIZE = 32;
```

```
    private static final int BLOCK_SIZE = 16;
    private static final int SHA_SIZE = 32;

            private byte[] matchPassword;
    private Mac hmac;
    private SecureRandom random;
    private MessageDigest digest;
    private IvParameterSpec ivSpec1;
    private SecretKeySpec aesKey1;
    private IvParameterSpec ivSpec2;
    private SecretKeySpec aesKey2;
}
```

FormUtama.java

```
import java.awt.FileDialog;
import java.awt.RenderingHints.Key;
import java.io.InputStream;
import java.io.OutputStream;
import javax.crypto.Cipher;
//import javax.crypto.SecretKeyFactory;
import javax.swing.JFrame;
import javax.swing.JPasswordField;
import javax.swing.ProgressMonitor;

import java.security.GeneralSecurityException;
import java.security.InvalidKeyException;
import java.io.BufferedOutputStream;
import java.util.Arrays;
import java.util.Enumeration;
import java.net.NetworkInterface;
import java.io.UnsupportedEncodingException;
import java.security.GeneralSecurityException;
import java.security.SecureRandom;
import java.security.MessageDigest;
import javax.crypto.spec.SecretKeySpec;
import javax.crypto.Mac;
import javax.crypto.spec.IvParameterSpec;

public class FormUtama extends javax.swing.JFrame {
```

```
private JFrame dummyFrame = new JFrame();
        private FileDialog fd;
        private String inputFile;
        private String passWord1;
        private String passWord2;
        private byte[] passByte;
        //private SecretKeyFactory keyFactory;
        private Key key;
        private JPasswordField pwd1;
        private JPasswordField pwd2;
        private Cipher cipher;
        private String outputFile;
        private InputStream in;
        private OutputStream out;
        private ProgressMonitor pMonitor;
        private int inputFileLength;
        private String inputfileName;
private int ERROR_MESSAGE;

public FormUtama() {
   initComponents();
}


@SuppressWarnings("unchecked")

private void initComponents() {
```

```
        jPanel1 = new javax.swing.JPanel();
        textboxEncrypt = new javax.swing.JTextField();
        jLabel1 = new javax.swing.JLabel();
        btbrowseEncrypt = new javax.swing.JButton();
        btEncrypt = new javax.swing.JButton();
        sizeenc = new javax.swing.JLabel();
        jPanel2 = new javax.swing.JPanel();
        jLabel2 = new javax.swing.JLabel();
        textboxDecrypt = new javax.swing.JTextField();
        btbrowseDcpt = new javax.swing.JButton();
        btDecrypt = new javax.swing.JButton();
        jMenuBar1 = new javax.swing.JMenuBar();
        jMenu1 = new javax.swing.JMenu();
        jMenuItem1 = new javax.swing.JMenuItem();
        jMenuItem2 = new javax.swing.JMenuItem();
        jMenuItem3 = new javax.swing.JMenuItem();
        jMenu2 = new javax.swing.JMenu();
        jMenuItem4 = new javax.swing.JMenuItem();
        jMenuItem5 = new javax.swing.JMenuItem();


setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
        setTitle("Algoritma AES Enkripsi and Dekripsi");
        setBounds(new java.awt.Rectangle(240, 180, 0, 0));
        setResizable(false);


jPanel1.setBorder(javax.swing.BorderFactory.createTitledBorder(javax.swing.B
orderFactory.createEtchedBorder(), "Encrypt File"));
```

```
textboxEncrypt.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        textboxEncryptActionPerformed(evt);
    }
});

jLabel1.setText("File :");

btbrowseEncrypt.setText("Browse");
btbrowseEncrypt.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        btbrowseEncryptActionPerformed(evt);
    }
});

btEncrypt.setText("Enkripsi");
btEncrypt.addChangeListener(new javax.swing.event.ChangeListener() {
    public void stateChanged(javax.swing.event.ChangeEvent evt) {
        btEncryptStateChanged(evt);
    }
});
btEncrypt.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        btEncryptActionPerformed(evt);
    }
});
```

```
        javax.swing.GroupLayout jPanel1Layout = new
javax.swing.GroupLayout(jPanel1);
        jPanel1.setLayout(jPanel1Layout);
        jPanel1Layout.setHorizontalGroup(

jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEAD
ING)
            .addGroup(jPanel1Layout.createSequentialGroup()
                .addContainerGap()
                .addComponent(jLabel1)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Align
ment.LEADING)
                    .addComponent(textboxEncrypt,
javax.swing.GroupLayout.DEFAULT_SIZE, 293, Short.MAX_VALUE)
                    .addComponent(sizeenc))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Align
ment.LEADING)
                    .addComponent(btbrowseEncrypt)
                    .addComponent(btEncrypt))
                .addContainerGap())
        );
        jPanel1Layout.setVerticalGroup(
```

```
jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEAD
ING)
            .addGroup(jPanel1Layout.createSequentialGroup()

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Align
ment.BASELINE)
                    .addComponent(jLabel1)
                    .addComponent(btbrowseEncrypt)
                    .addComponent(textboxEncrypt,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))

.addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Align
ment.LEADING)
                    .addGroup(jPanel1Layout.createSequentialGroup()
                        .addGap(18, 18, 18)
                        .addComponent(btEncrypt))
                    .addGroup(jPanel1Layout.createSequentialGroup()
                        .addGap(8, 8, 8)
                        .addComponent(sizeenc)))
                .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE))
        );


jPanel2.setBorder(javax.swing.BorderFactory.createTitledBorder(javax.swing.B
orderFactory.createEtchedBorder(), "Decrypt File"));
```

```
        jLabel2.setText("File :");

        btbrowseDcpt.setText("Browse");
        btbrowseDcpt.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                btbrowseDcptActionPerformed(evt);
            }
        });

        btDecrypt.setText("Dekripsi");
        btDecrypt.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                btDecryptActionPerformed(evt);
            }
        });

        javax.swing.GroupLayout jPanel2Layout = new
javax.swing.GroupLayout(jPanel2);
        jPanel2.setLayout(jPanel2Layout);
        jPanel2Layout.setHorizontalGroup(

jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEAD
ING)
            .addGroup(jPanel2Layout.createSequentialGroup()
                .addContainerGap()
                .addComponent(jLabel2)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
```

```
                .addComponent(textboxDecrypt,
javax.swing.GroupLayout.PREFERRED_SIZE, 291,
javax.swing.GroupLayout.PREFERRED_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Align
ment.LEADING)
                    .addComponent(btbrowseDcpt)
                    .addComponent(btDecrypt))
                .addContainerGap(12, Short.MAX_VALUE))
        );
        jPanel2Layout.setVerticalGroup(

jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEAD
ING)
            .addGroup(jPanel2Layout.createSequentialGroup()

.addGroup(jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Align
ment.BASELINE)
                    .addComponent(jLabel2)
                    .addComponent(textboxDecrypt,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(btbrowseDcpt))
                .addGap(18, 18, 18)
                .addComponent(btDecrypt)
```

```
            .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE))
        );

        jMenu1.setText("File");


jMenuItem1.setAccelerator(javax.swing.KeyStroke.getKeyStroke(java.awt.even
t.KeyEvent.VK_E, java.awt.event.InputEvent.CTRL_MASK));
        jMenuItem1.setText("Buka File Untuk Enkripsi");
        jMenuItem1.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jMenuItem1ActionPerformed(evt);
            }
        });
        jMenu1.add(jMenuItem1);


jMenuItem2.setAccelerator(javax.swing.KeyStroke.getKeyStroke(java.awt.even
t.KeyEvent.VK_D, java.awt.event.InputEvent.CTRL_MASK));
        jMenuItem2.setText("Buka File untuk Dekripsi");
        jMenuItem2.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jMenuItem2ActionPerformed(evt);
            }
        });
        jMenu1.add(jMenuItem2);
```

```
jMenuItem3.setAccelerator(javax.swing.KeyStroke.getKeyStroke(java.awt.even
t.KeyEvent.VK_Q, java.awt.event.InputEvent.CTRL_MASK));
        jMenuItem3.setText("Keluar");
        jMenuItem3.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jMenuItem3ActionPerformed(evt);
            }
        });
        jMenu1.add(jMenuItem3);

        jMenuBar1.add(jMenu1);

        jMenu2.setText("About");
        jMenu2.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jMenu2ActionPerformed(evt);
            }
        });


            jMenuItem5.setAccelerator(javax.swing.KeyStroke.getKeyStroke(java
.awt.event.KeyEvent.VK_F1, 0));
        jMenuItem5.setText("About Me");
        jMenuItem5.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jMenuItem5ActionPerformed(evt);
            }
        });
```

```
        jMenu2.add(jMenuItem5);

        jMenuBar1.add(jMenu2);

        setJMenuBar(jMenuBar1);

        javax.swing.GroupLayout layout = new
javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TR
AILING)
                    .addComponent(jPanel2,
javax.swing.GroupLayout.Alignment.LEADING,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
                    .addComponent(jPanel1,
javax.swing.GroupLayout.Alignment.LEADING,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
                .addContainerGap())
        );
        layout.setVerticalGroup(
```

```
layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()
                .addComponent(jPanel1,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(jPanel2,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addContainerGap(22, Short.MAX_VALUE))
        );

        pack();
    }

private void textboxEncryptActionPerformed(java.awt.event.ActionEvent evt) {
}

private void btbrowseEncryptActionPerformed(java.awt.event.ActionEvent evt)
{
    Enkripsi dec = new Enkripsi();
    Enkripsi.openFile();
    textboxEncrypt.toString();
    textboxEncrypt.setText(Enkripsi.inputFile);
```

```
}

private void btbrowseDcptActionPerformed(java.awt.event.ActionEvent evt) {
    Dekripsi dec = new Dekripsi();
    Dekripsi.openFile();
    textboxDecrypt.toString();
    textboxDecrypt.setText(Dekripsi.inputFile);

}

private void btEncryptActionPerformed(java.awt.event.ActionEvent evt) {
 Enkripsi enkrip = new Enkripsi();
 Enkripsi.aksi();

}

private void btEncryptStateChanged(javax.swing.event.ChangeEvent evt) {
}

private void jMenuItem3ActionPerformed(java.awt.event.ActionEvent evt) {
    System.exit(0);
}

private void btDecryptActionPerformed(java.awt.event.ActionEvent evt) {
    Dekripsi dekrip = new Dekripsi();
    Dekripsi.aksi();
}
```

```
private void jMenuItem1ActionPerformed(java.awt.event.ActionEvent evt) {
  Enkripsi dec = new Enkripsi();
    Enkripsi.openFile();
    textboxEncrypt.toString();
    textboxEncrypt.setText(Enkripsi.inputFile);
}

private void jMenuItem2ActionPerformed(java.awt.event.ActionEvent evt) {
Dekripsi dec = new Dekripsi();
    Dekripsi.openFile();
    textboxDecrypt.toString();
    textboxDecrypt.setText(Dekripsi.inputFile);
}

private void jMenu2ActionPerformed(java.awt.event.ActionEvent evt) {
}

private void jMenuItem5ActionPerformed(java.awt.event.ActionEvent evt) {
Tentang about = new Tentang();
    about.show();

}



    public static void main(String args[]) {
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new FormUtama().setVisible(true);
```

```
        }
    });
}

    private javax.swing.JButton btDecrypt;
    private javax.swing.JButton btEncrypt;
    private javax.swing.JButton btbrowseDcpt;
    private javax.swing.JButton btbrowseEncrypt;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JMenu jMenu1;
    private javax.swing.JMenu jMenu2;
    private javax.swing.JMenuBar jMenuBar1;
    private javax.swing.JMenuItem jMenuItem1;
    private javax.swing.JMenuItem jMenuItem2;
    private javax.swing.JMenuItem jMenuItem3;
    private javax.swing.JMenuItem jMenuItem4;
    private javax.swing.JMenuItem jMenuItem5;
    private javax.swing.JPanel jPanel1;
    private javax.swing.JPanel jPanel2;
    private javax.swing.JLabel sizeenc;
    private javax.swing.JTextField textboxDecrypt;
    private javax.swing.JTextField textboxEncrypt;

}
```

Tentang.java

```
public class Tentang extends javax.swing.JFrame {

    public Tentang() {
        initComponents();
    }

    @SuppressWarnings("unchecked")

    private void initComponents() {

        jScrollPane2 = new javax.swing.JScrollPane();
        jTextArea2 = new javax.swing.JTextArea();


setDefaultCloseOperation(javax.swing.WindowConstants.DISPOSE_ON_CLO
SE);
        setTitle("Tentang Aplikasi");
        setBounds(new java.awt.Rectangle(255, 240, 0, 0));
        setResizable(false);


jTextArea2.setBackground(javax.swing.UIManager.getDefaults().getColor("Co
mboBox.buttonBackground"));
        jTextArea2.setColumns(20);
        jTextArea2.setEditable(false);
        jTextArea2.setForeground(new java.awt.Color(0, 153, 255));
```

```
        jTextArea2.setRows(5);
        jTextArea2.setText("Program Enkripsi AES \nCreative By : Harapan
(2005-81-053) \nTeknik Informatika - Universitas Indonusa Esa Unggul
\nCopyright(c)2011");
        jScrollPane2.setViewportView(jTextArea2);

        javax.swing.GroupLayout layout = new
javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()
                .addComponent(jScrollPane2,
javax.swing.GroupLayout.PREFERRED_SIZE, 422,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE))
        );
        layout.setVerticalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()
                .addComponent(jScrollPane2,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
```

```
                .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE))
        );

        pack();
    }

    public static void main(String args[]) {
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new Tentang().setVisible(true);
            }
        });
    }

    private javax.swing.JScrollPane jScrollPane2;
    private javax.swing.JTextArea jTextArea2;

}
```