



INSTITUTO POLITÉCNICO NACIONAL
ESCUELA SUPERIOR DE CÓMPUTO



Cryptography

"DES/AES"

Abstact

The Data Encryption Standard is a symmetric-key algorithm for the encryption of electronic data. Although insecure, it was highly influential in the advancement of modern cryptography.

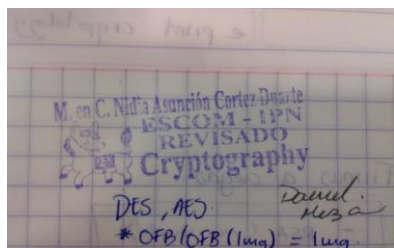
By:

Meza Martínez Luis Daniel

Professor:

MSc. NIDIA ASUNCIÓN CORTEZ DUARTE

June 2018



Index

Contenido

Introduction:	1
Literature review:.....	1
Software (libraries, packages, tools):.....	3
Procedure:.....	4
Results	4
Discussion:.....	10
Conclusions:	10
References:.....	10
Code.....	10

Introduction:

To carry out this practice, we need to know how to use BMP images, which can be read because, as the name says, they are bitmaps

The BMP file format, also known as bitmap image file or device independent bitmap (DIB) file format or simply a bitmap, is a raster graphics image file format used to store bitmap digital images, independently of the display device (such as a graphics adapter), especially on Microsoft Windows and OS operating systems.

The BMP file format is capable of storing two-dimensional digital images both monochrome and color, in various color depths, and optionally with data compression, alpha channels, and color profiles. The Windows Metafile (WMF) specification covers the BMP file format.

In the 1973, the National Bureau of Standards, inter to become the NATIONAL Institute of Standards and Tecnology (NIST), issued a public request seeking a cryptographic algorithm to become a national standar. In 1975 NBS released DES, as well as a free license for its use, and in 1977 NBS made it the official data encryption standard.

DES has been used extensively in electronic commerce for example in the banking industry. If two banks want to exchange data, the first use a public key method such as RSA to transmit a key for DES, then they use DES for transmitting the data. It has the advantage of being very fast and reasonably secure.

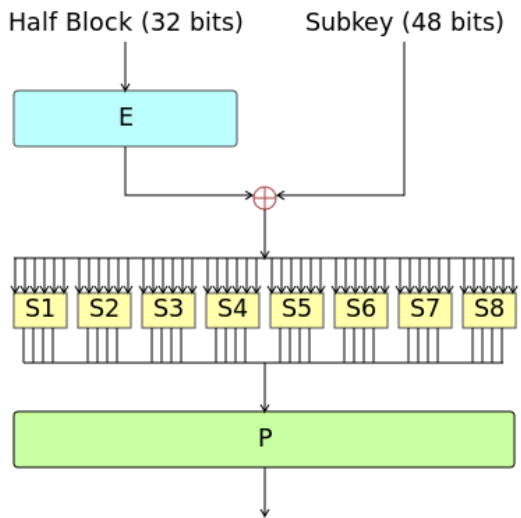
Literature review:

DES is a symmetric block cipher (shared secret key), with a key length of 56-bits. Published as the Federal Information Processing Standards (FIPS) 46 standard in 1977, DES was officially withdrawn in 2005 although NIST has approved Triple DES (3DES) through 2030 for sensitive government information.

The federal government originally developed DES encryption over 35 years ago to provide cryptographic security for all government communications. The idea was to ensure government systems all used the same, secure standard to facilitate interconnectivity.

To show that the DES was inadequate and should not be used in important systems anymore, a series of challenges were sponsored to see how long it would take to decrypt a message. Two organizations played key roles in breaking DES: distributed.net and the Electronic Frontier Foundation (EFF).

Des description



DES is the archetypal block cipher—an algorithm that takes a fixed-length string of plaintext bits and transforms it through a series of complicated operations into another ciphertext bitstring of the same length. In the case of DES, the block size is 64 bits. DES also uses a key to customize the transformation, so that decryption can supposedly only be performed by those who know the particular key used to encrypt. The key ostensibly consists of 64 bits; however, only 56 of these are actually used by the algorithm. Eight bits are used solely for checking parity, and are thereafter discarded. Hence the effective key length is 56 bits.

The key is nominally stored or transmitted as 8 bytes, each with odd parity. According to ANSI X3.92-1981 (Now, known as ANSI INCITS 92-1981), section 3.5:

One bit in each 8-bit byte of the KEY may be utilized for error detection in key generation, distribution, and storage. Bits 8, 16, ..., 64 are for use in ensuring that each byte is of odd parity.

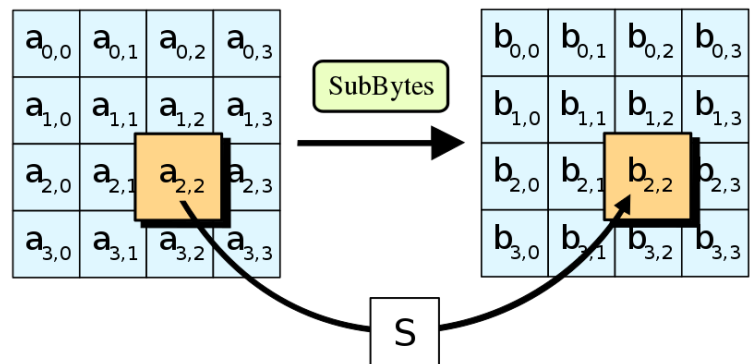
Like other block ciphers, DES by itself is not a secure means of encryption, but must instead be used in a mode of operation. FIPS-81 specifies several modes for use with DES. Further comments on the usage of DES are contained in FIPS-74.

Decryption uses the same structure as encryption, but with the keys used in reverse order. (This has the advantage that the same hardware or software can be used in both directions).

AES

The Advanced Encryption Standard (AES), also known by its original name Rijndael, is a specification for the encryption of electronic data established by the U.S. National Institute of Standards and Technology (NIST) in 2001.

AES is based on a design principle known as a substitution-permutation network, a combination of both substitution and permutation, and is fast in both software and hardware. Unlike its predecessor DES, AES does not use a Feistel network. AES is a variant of Rijndael which has a fixed block size of 128 bits, and a key size of 128, 192, or 256 bits. By contrast, the Rijndael specification per se



is specified with block and key sizes that may be any multiple of 32 bits, with a minimum of 128 and a maximum of 256 bits.

AES operates on a 4×4 column-major order matrix of bytes, termed the state, although some versions of Rijndael have a larger block size and have additional columns in the state. Most AES calculations are done in a particular finite field.

Software (libraries, packages, tools):



For the realization of this practice the language that I used to carry out this practice was Java, and the IDE that I use netbeans, because I needed a graphic interface and it was easier for me to develop in netbeans.

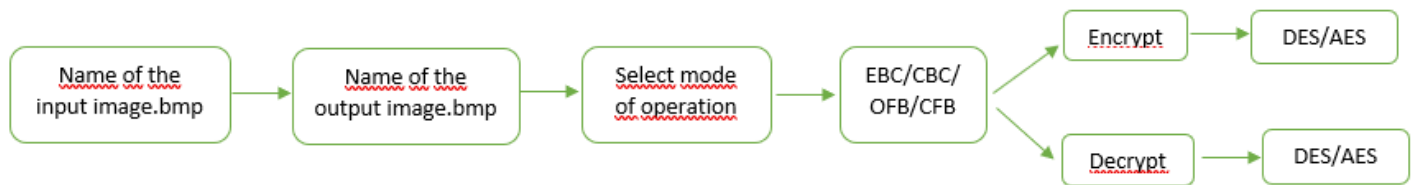
I thought to present this practice in python, however I still need to study the graphic part of python, in the future I hope to have a better graphic interface.



Libraries:

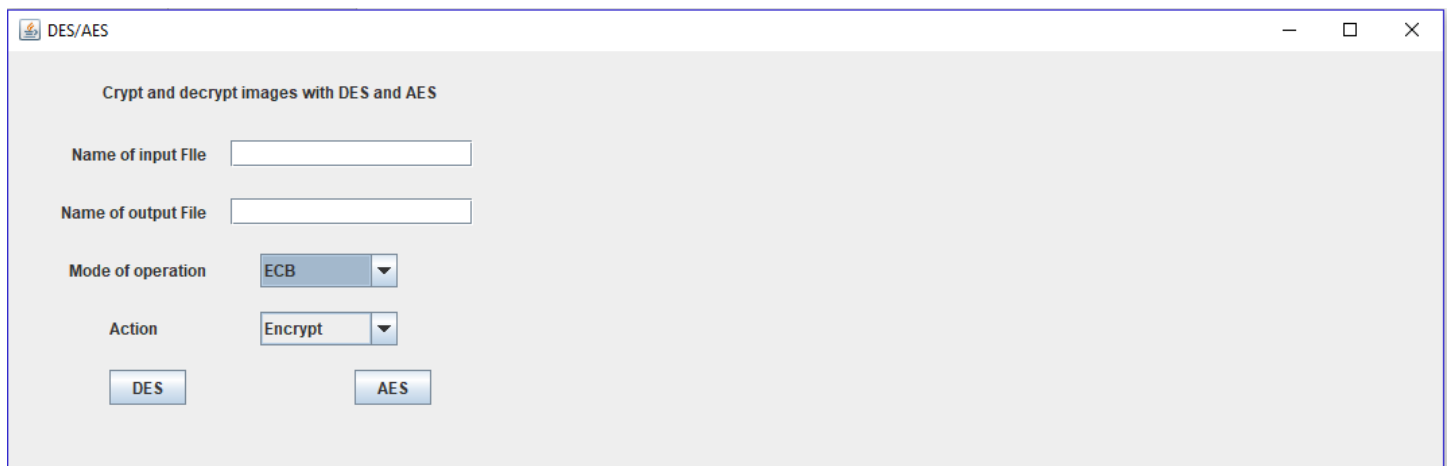
```
1. package desaes;
2.
3. import java.awt.Image;
4. import java.awt.image.BufferedImage;
5. import java.io.ByteArrayInputStream;
6. import java.io.ByteArrayOutputStream;
7. import java.io.File;
8. import java.io.IOException;
9. import java.io.InputStream;
10. import java.security.NoSuchAlgorithmException;
11. import java.security.Security;
12. import java.security.spec.AlgorithmParameterSpec;
13. import java.util.logging.Level;
14. import java.util.logging.Logger;
15. import javax.crypto.BadPaddingException;
16. import javax.crypto.Cipher;
17. import javax.crypto.IllegalBlockSizeException;
18. import javax.crypto.KeyGenerator;
19. import javax.crypto.SecretKey;
20. import javax.crypto.spec.IvParameterSpec;
21. import javax.crypto.spec.SecretKeySpec;
22. import javax.imageio.ImageIO;
23. import javax.swing.ImageIcon;
24. import javax.swing.JOptionPane;
```

Procedure:

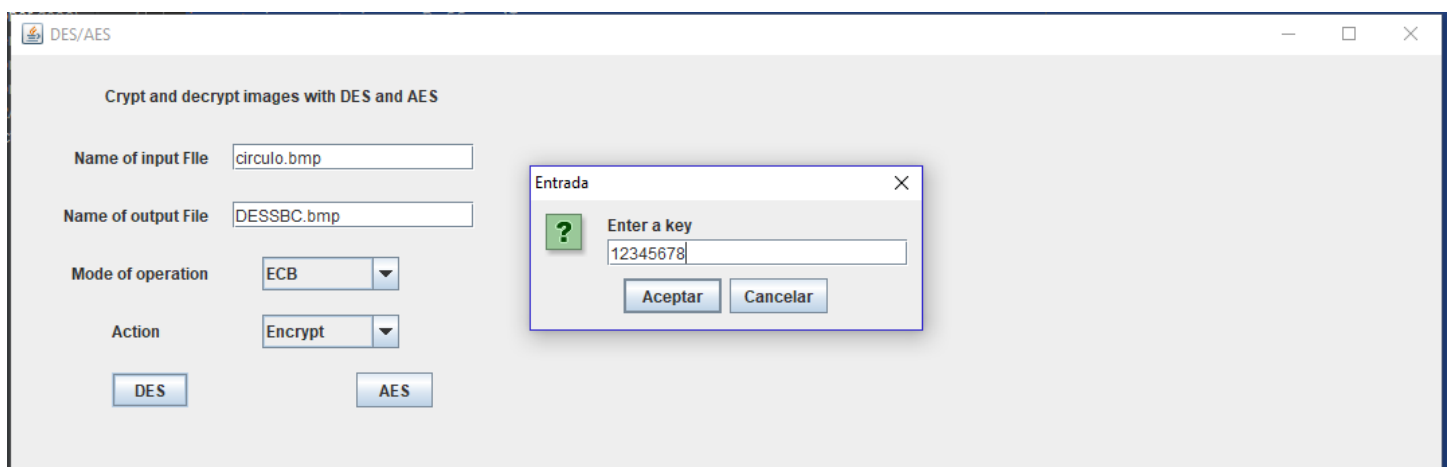


Results

My interface was like this.



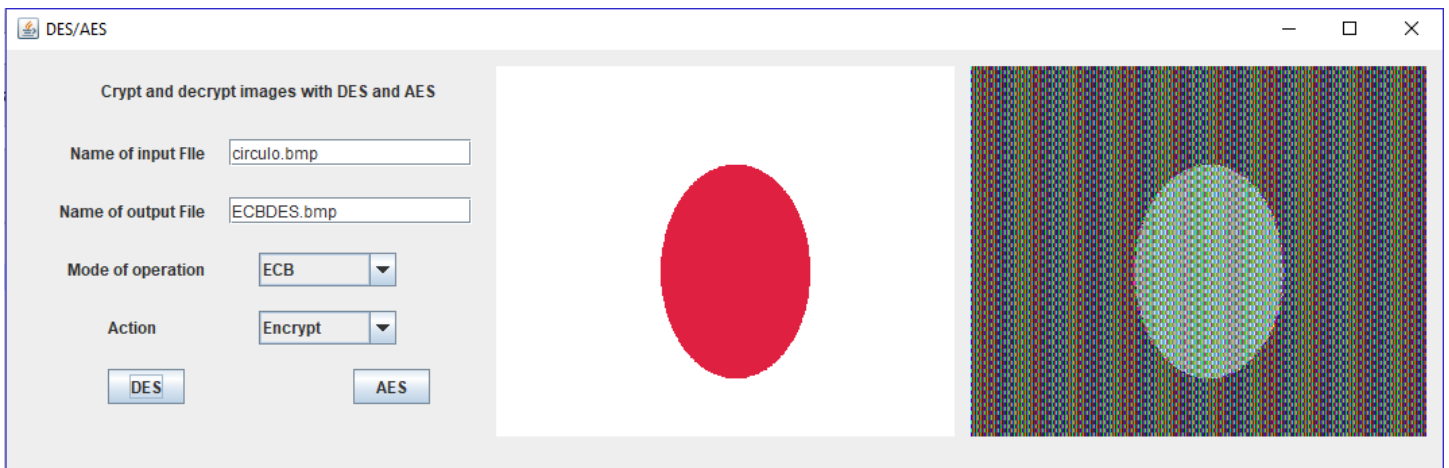
We need to write the name of our image and put the name of the output image ends with .bmp, we need to choose a mode of operation, like ECB, CBC, OFB or CFB and later we choose if we want to encrypt or decrypt the image, and final! We need to choose if we want the AES or DES.



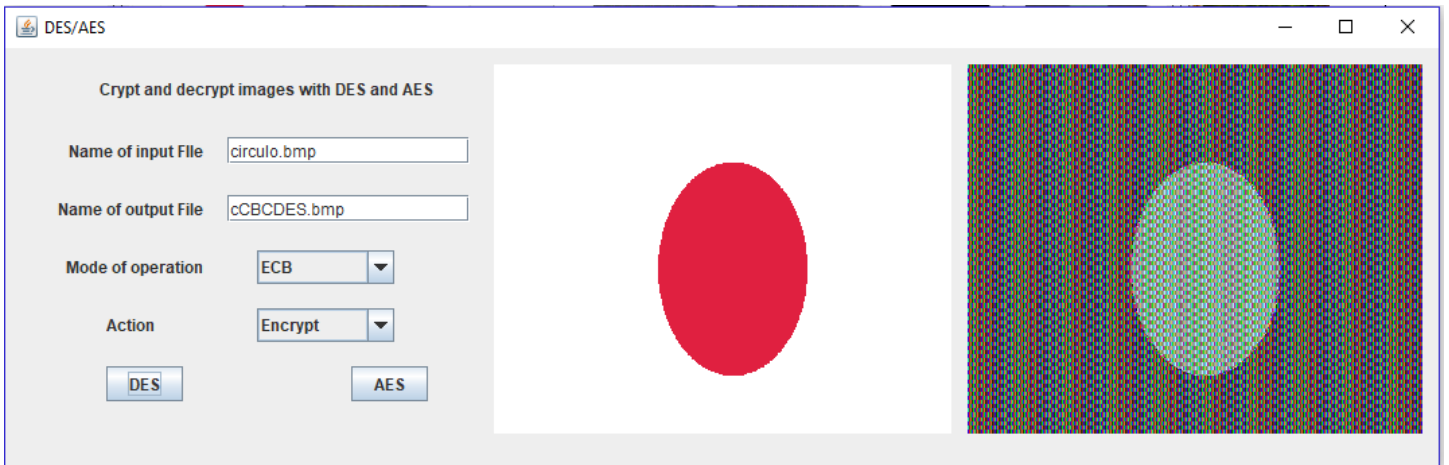
For later we need to write and a key, to decrypt later.

In the example we use for key 12345678

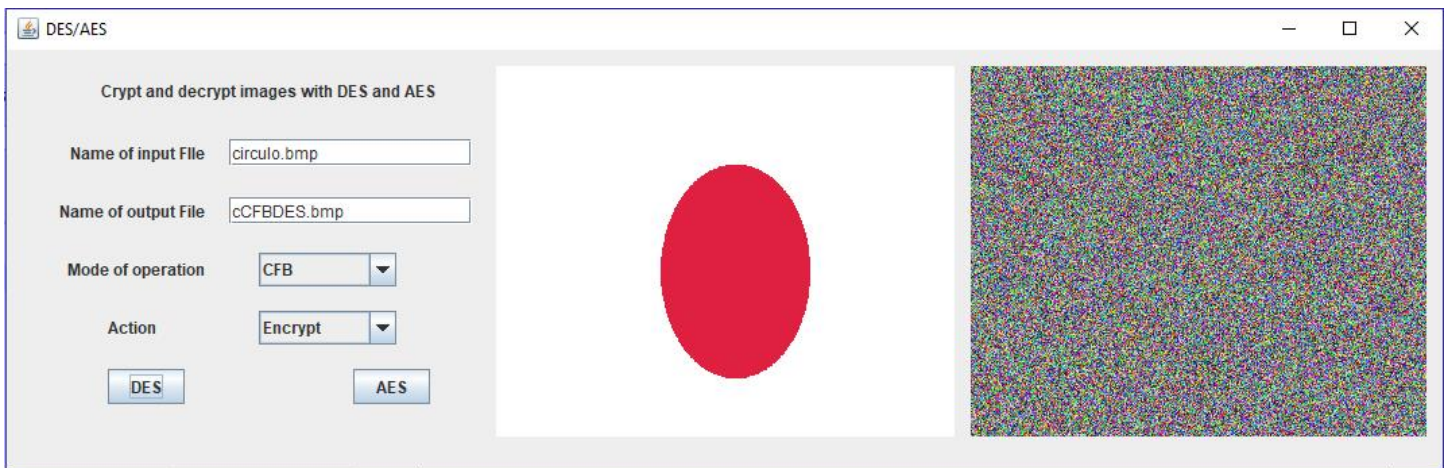
With mode of operation ECB and using DES



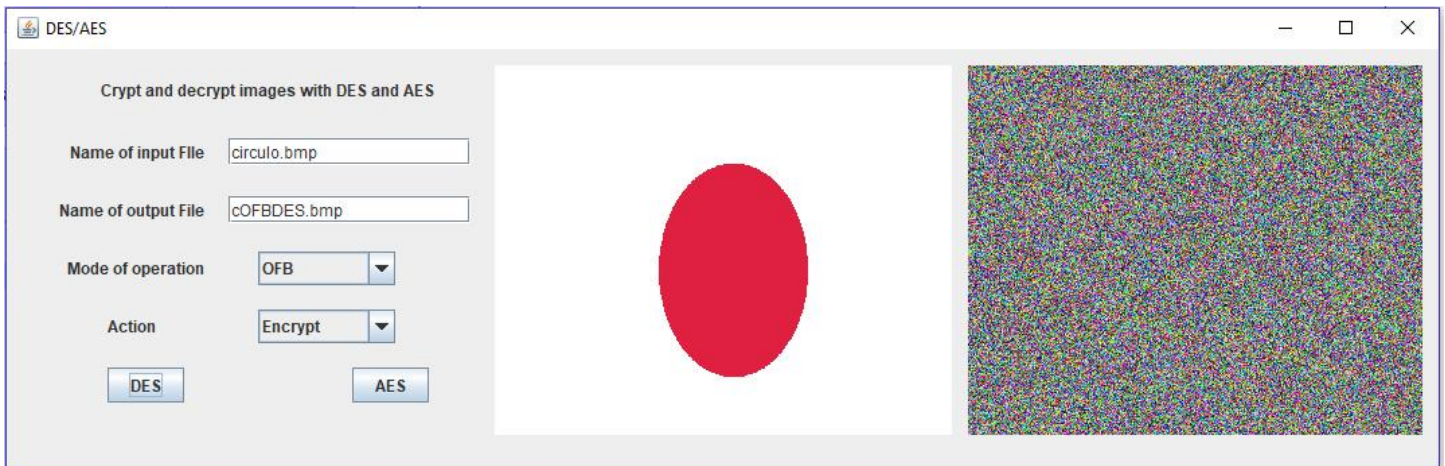
With mode of operation CBC and using DES



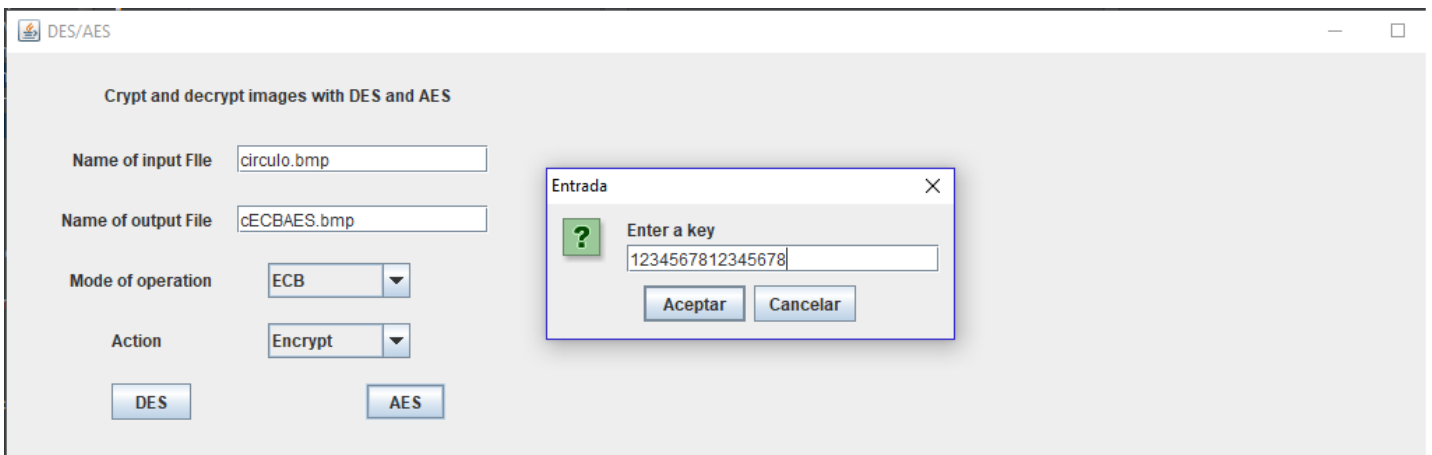
With mode of operation CFB and using DES



With mode of operation OFB and using DES

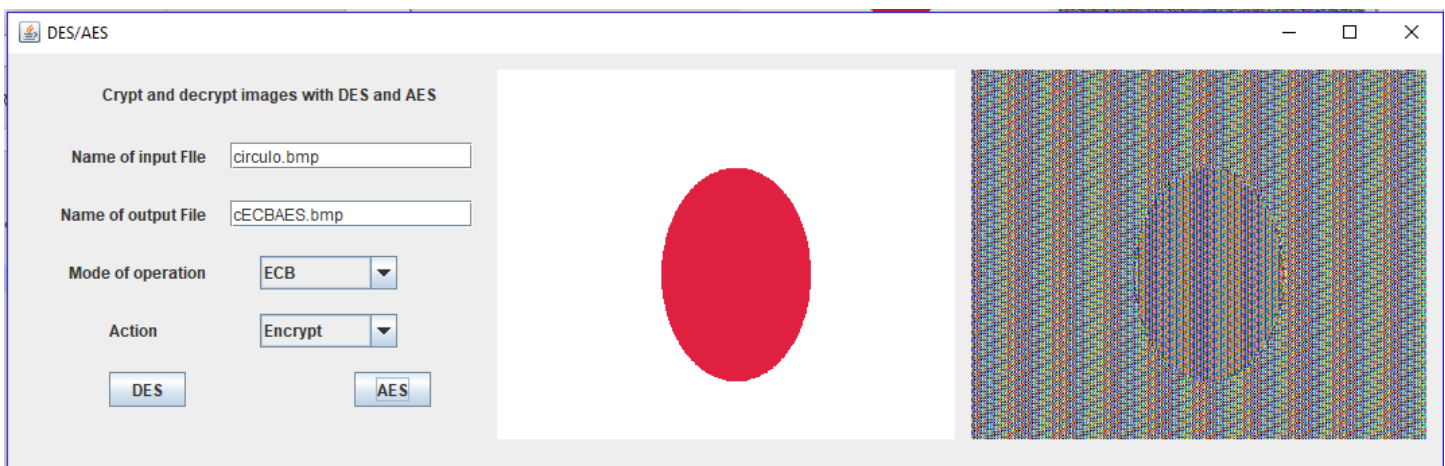


To use AES the length of our key is of 16.

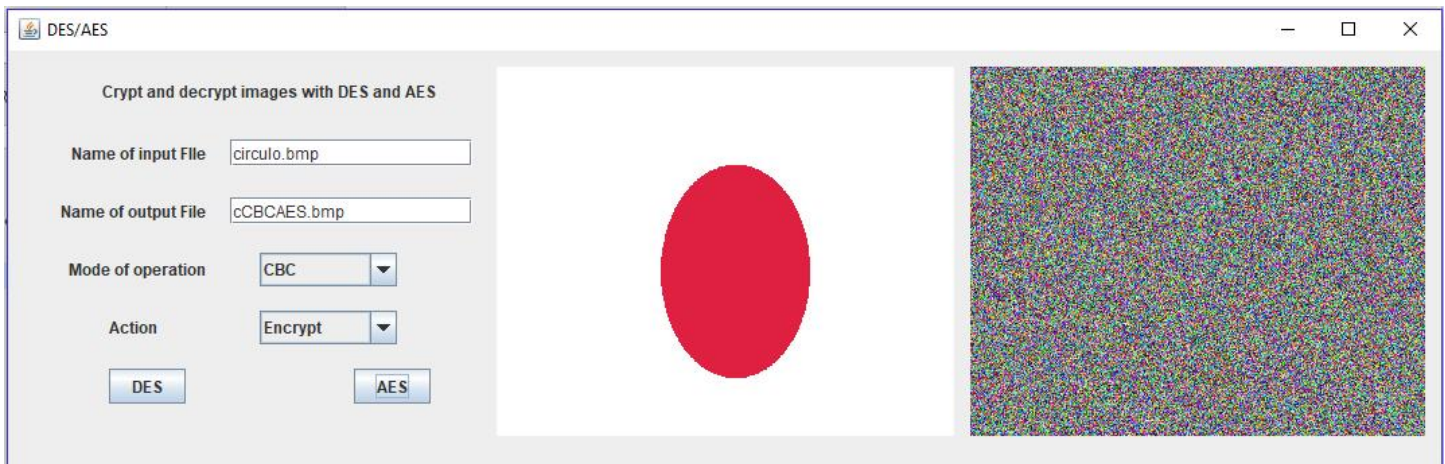


In the example we use for key 1234567812345678

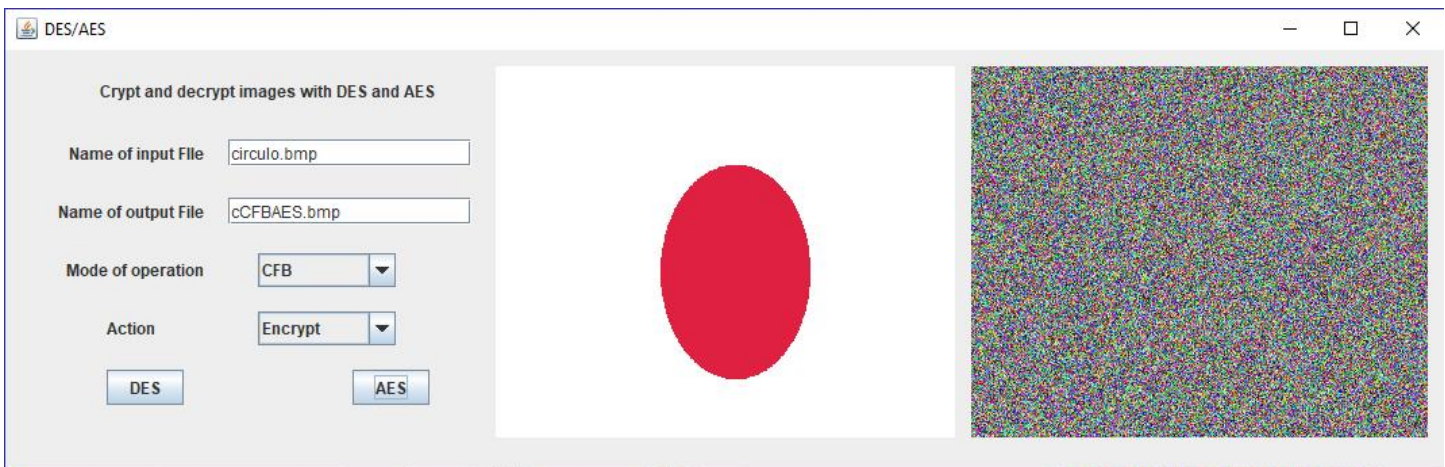
With mode of operation ECB and using AES



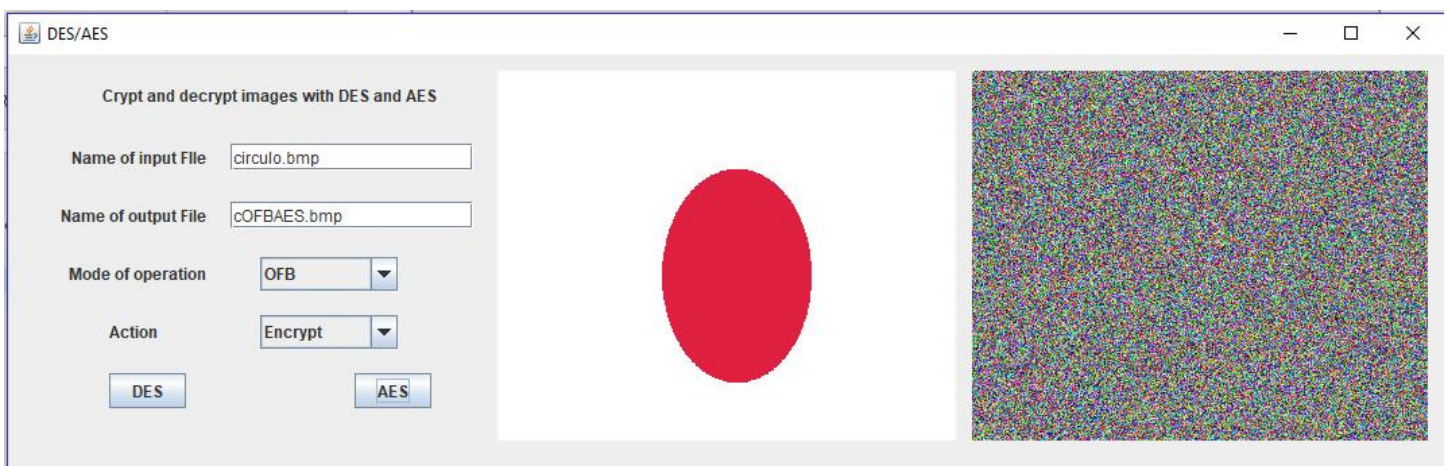
With mode of operation CBC and using AES

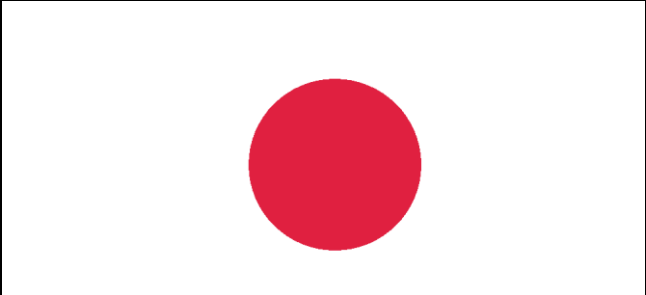

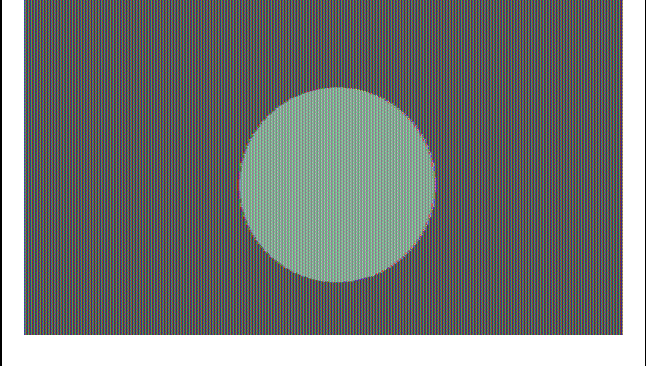
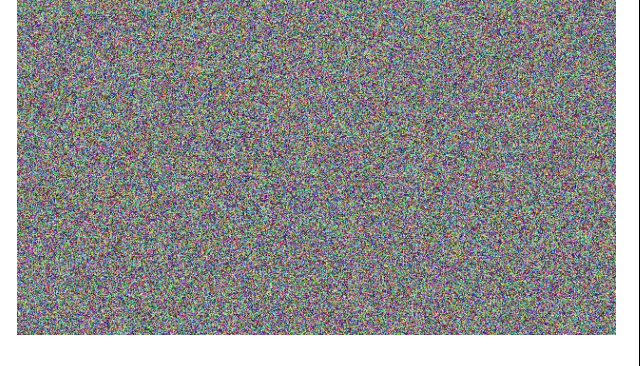
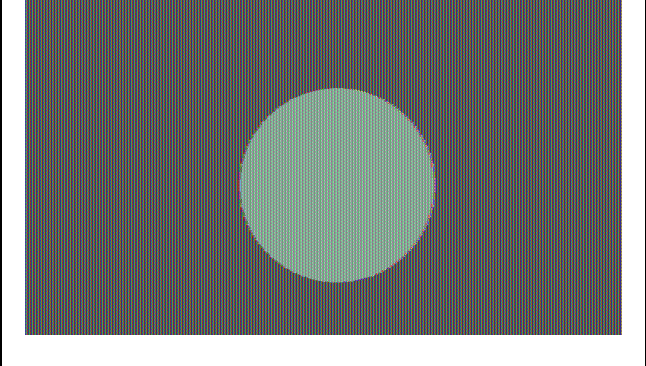
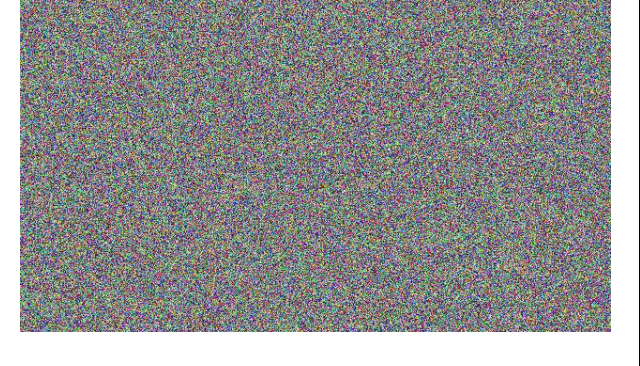
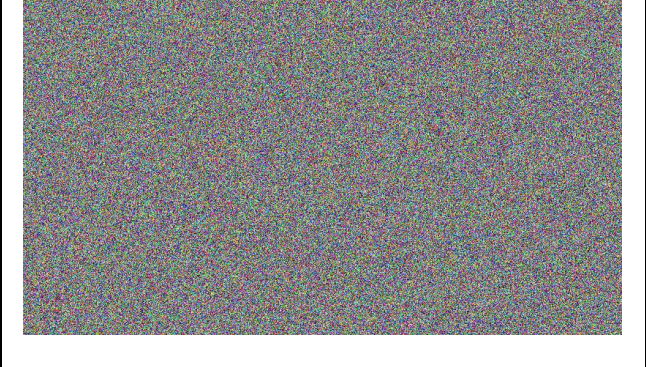
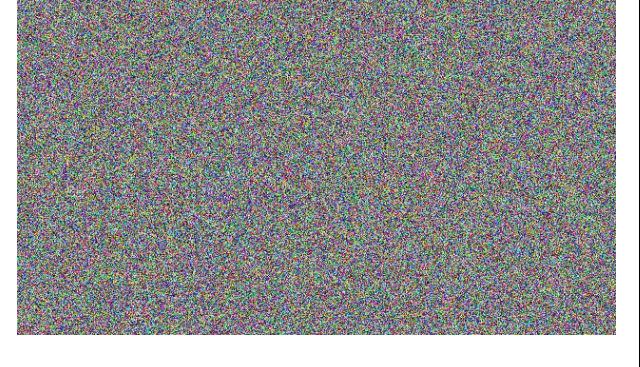
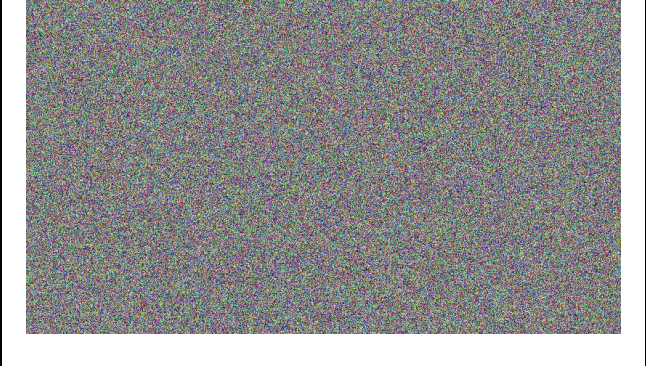
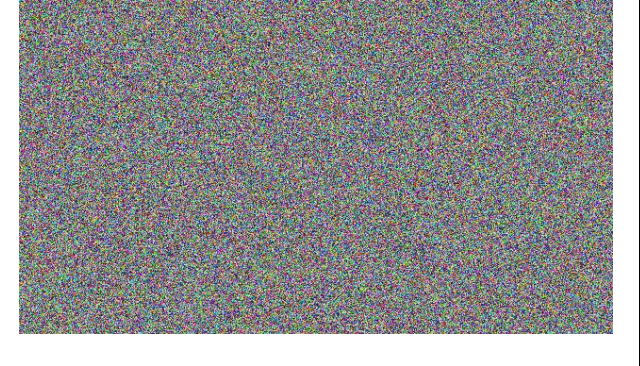




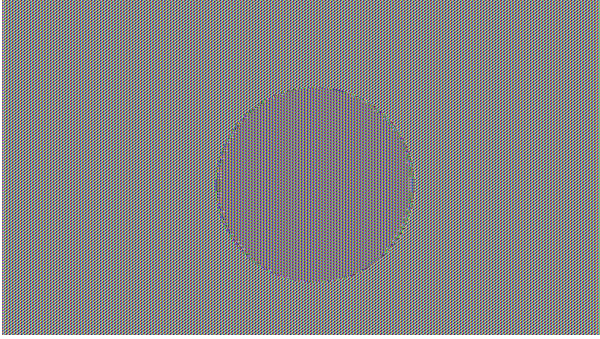


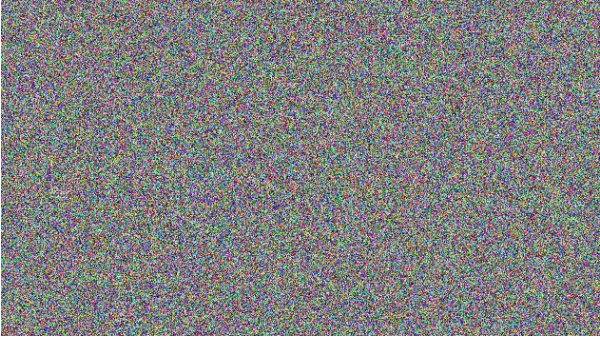
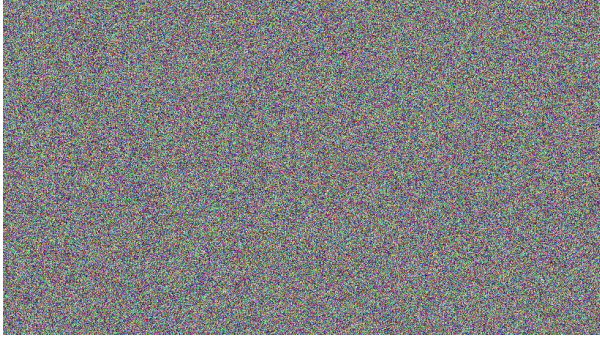

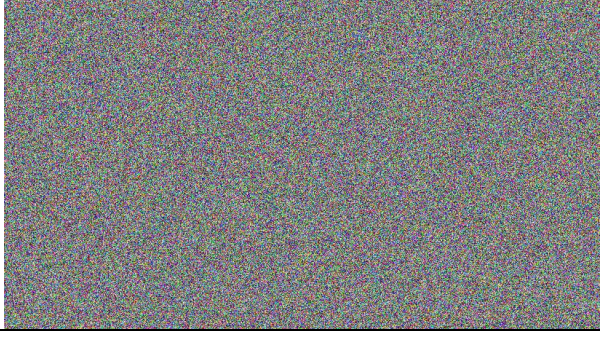
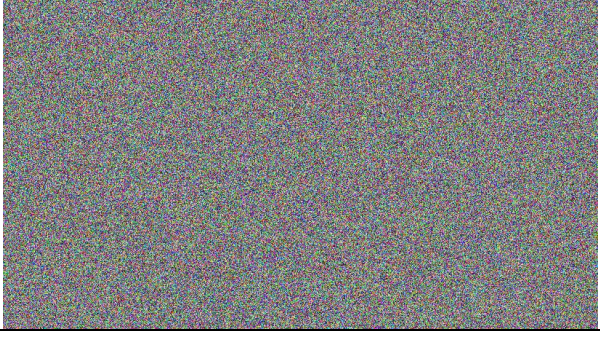
With mode of operation CFB and using AES



With mode of operation OFB and using AES



DES		
ORIGINAL		
ECB		
CBC		
CFB		
OFB		

AES		
ORIGINAL		
ECB		
CBC		
CFB		
OFB		

Discussion:

With the use of AES or DES we know more about algorithms of cipher, we know all of this algorithm help us to cipher the information, the difference between AES and DES is in the year was developed, and the key length.

	DES	AES
Developed	1977	2000
Key Length	56 bits	128, 192, or 256 bits
Cipher Type	Symmetric block cipher	Symmetric block cipher
Block Size	64 bits	128 bits
Security	Proven inadequate	Considered secure

Conclusions:

To make this practice was very hard, because I was trying for a lot of time to cipher the BMP images, sometimes, I think the problem was in the header of the BMP images.

References:

Knudsen, J. (1998). Java cryptography. Cambridge: O'Reilly.

Konheim, A. (2007). Computer security and cryptography. New York, N.Y.: Wiley.

Introduction to Cryptography with Coding Theory, 2nd edition By Wade Trappe and Lawrence C. Washington

Code

```
1. package desaes;
2.
3. import java.awt.Image;
4. import java.awt.image.BufferedImage;
5. import java.io.ByteArrayInputStream;
6. import java.io.ByteArrayOutputStream;
7. import java.io.File;
8. import java.io.IOException;
```

```

9. import java.io.InputStream;
10. import java.security.NoSuchAlgorithmException;
11. import java.security.Security;
12. import java.security.spec.AlgorithmParameterSpec;
13. import java.util.logging.Level;
14. import java.util.logging.Logger;
15. import javax.crypto.BadPaddingException;
16. import javax.crypto.Cipher;
17. import javax.crypto.IllegalBlockSizeException;
18. import javax.crypto.KeyGenerator;
19. import javax.crypto.SecretKey;
20. import javax.crypto.spec.IvParameterSpec;
21. import javax.crypto.spec.SecretKeySpec;
22. import javax.imageio.ImageIO;
23. import javax.swing.ImageIcon;
24. import javax.swing.JOptionPane;
25. /**
26.  *
27.  * @author ldmm
28.  */
29. public class desaes extends javax.swing.JFrame
30. {
31.
32.     private static Cipher enc;
33.     private static Cipher dec;
34.     //InitializationVector DES
35.     private static final byte[] iV = { 11, 22, 33, 44, 99, 88, 77, 66 };
36.     //InitializationVector AES
37.     private static final byte[] iVAES = { 11, 22, 33, 44, 55, 66, 77, 88, 99, 11, 22, 33, 44, 55, 66,
77 };
38.     //Header size
39.     private static final int headerZ = 54;
40.     //Llave simétrica
41.     private SecretKey simmetricKey;
42.     //Llave simétrica AES
43.     private SecretKey simmetricKeyAES;
44.     private AlgorithmParameterSpec algorithmParameterSpec;
45.     private AlgorithmParameterSpec algorithmParameterSpecAES;
46.     private String fileInput;
47.     private String fileOutput;
48.     public desaes() throws NoSuchAlgorithmException
49.     {
50.         initComponents();
51.
52.         setTitle("DES/AES");
53.         //Generation of keys
54.         simmetricKey = KeyGenerator.getInstance("DES").generateKey();
55.         System.out.println(simmetricKey);
56.         simmetricKeyAES = KeyGenerator.getInstance("AES").generateKey();
57.         //Initialization of vectors
58.         algorithmParameterSpec = new IvParameterSpec(iV);
59.         algorithmParameterSpecAES = new IvParameterSpec(iVAES);
60.         this.setLocationRelativeTo(null);
61.     }
62.
63.
64.
65.     @SuppressWarnings("unchecked")
66.     // <editor-fold defaultstate="collapsed" desc="Generated Code">
67.     private void initComponents() {
68.
69.         jLabel1 = new javax.swing.JLabel();
70.         jLabel5 = new javax.swing.JLabel();
71.         jTextField2 = new javax.swing.JTextField();
72.         jLabel2 = new javax.swing.JLabel();

```

```

73.      jTextField1 = new javax.swing.JTextField();
74.      jLabel3 = new javax.swing.JLabel();
75.      jComboBox1 = new javax.swing.JComboBox();
76.      jComboBox2 = new javax.swing.JComboBox();
77.      jLabel4 = new javax.swing.JLabel();
78.      jButton1 = new javax.swing.JButton();
79.      jButton2 = new javax.swing.JButton();
80.      jLabel6 = new javax.swing.JLabel();
81.      jLabel7 = new javax.swing.JLabel();
82.
83.      setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
84.
85.      jLabel1.setText("Crypt and decrypt images with DES and AES");
86.
87.      jLabel5.setText("Name of output File");
88.
89.      jLabel2.setText("Name of input FILE");
90.
91.      jTextField1.addActionListener(new java.awt.event.ActionListener() {
92.          public void actionPerformed(java.awt.event.ActionEvent evt) {
93.              jTextField1ActionPerformed(evt);
94.          }
95.      });
96.
97.      jLabel3.setText("Mode of operation");
98.
99.      jComboBox1.setModel(new javax.swing.DefaultComboBoxModel(new String[] { "ECB", "CBC", "CFB",
"OFB" }));
100.
101.      jComboBox2.setModel(new javax.swing.DefaultComboBoxModel(new String[] { "Encrypt", "De
crypt" }));
102.      jComboBox2.setCursor(new java.awt.Cursor(java.awt.Cursor.DEFAULT_CURSOR));
103.      jComboBox2.addActionListener(new java.awt.event.ActionListener() {
104.          public void actionPerformed(java.awt.event.ActionEvent evt) {
105.              jComboBox2ActionPerformed(evt);
106.          }
107.      });
108.
109.      jLabel4.setText("Action");
110.
111.      jButton1.setText("DES");
112.      jButton1.addActionListener(new java.awt.event.ActionListener() {
113.          public void actionPerformed(java.awt.event.ActionEvent evt) {
114.              jButton1ActionPerformed(evt);
115.          }
116.      });
117.
118.      jButton2.setText("AES");
119.      jButton2.addActionListener(new java.awt.event.ActionListener() {
120.          public void actionPerformed(java.awt.event.ActionEvent evt) {
121.              jButton2ActionPerformed(evt);
122.          }
123.      });
124.
125.      javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
126.      getContentPane().setLayout(layout);
127.      layout.setHorizontalGroup(
128.          layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
129.              .addGroup(layout.createSequentialGroup()
130.                  .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
131.                      .addGroup(layout.createSequentialGroup()
132.                          .addGap(39, 39, 39)
133.                          .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment

```



```

134.                .addComponent(jLabel13)
135.                .addComponent(jLabel12)
136.                .addComponent(jLabel15)
137.                .addGroup(layout.createSequentialGroup())
138.                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.A
    lignment.LEADING)
139.                    .addComponent(jButton1)
140.                    .addComponent(jLabel14))
141.                .addGap(15, 15, 15)))
142.                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment
    .LEADING)
143.                    .addGroup(layout.createSequentialGroup())
144.                    .addGap(40, 40, 40)
145.                    .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.A
    lignment.LEADING)
146.                        .addComponent(jComboBox2, javax.swing.GroupLayout.PREFERRE
    D_SIZE, 102, javax.swing.GroupLayout.PREFERRED_SIZE)
147.                        .addComponent(jComboBox1, javax.swing.GroupLayout.PREFERRE
    D_SIZE, 102, javax.swing.GroupLayout.PREFERRED_SIZE)))
148.                    .addGroup(layout.createSequentialGroup())
149.                    .addGap(18, 18, 18)
150.                    .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.A
    lignment.LEADING)
151.                        .addGroup(layout.createSequentialGroup())
152.                        .addGap(92, 92, 92)
153.                        .addComponent(jButton2))
154.                    .addGroup(layout.createParallelGroup(javax.swing.GroupLayo
    ut.Alignment.LEADING, false)
155.                        .addComponent(jTextField1)
156.                        .addComponent(jTextField2, javax.swing.GroupLayout.PRE
    FERRED_SIZE, 180, javax.swing.GroupLayout.PREFERRED_SIZE))))))
157.                .addGroup(layout.createSequentialGroup())
158.                .addGap(70, 70, 70)
159.                .addComponent(jLabel11)))
160.                .addGap(18, 18, 18)
161.                .addComponent(jLabel6, javax.swing.GroupLayout.DEFAULT_SIZE, 340, Short.MAX_VA
    LUE)
162.                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
163.                .addComponent(jLabel7, javax.swing.GroupLayout.PREFERRED_SIZE, 338, javax.swin
    g.GroupLayout.PREFERRED_SIZE)
164.                .addContainerGap()
165.            );
166.            layout.setVerticalGroup(
167.                layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
168.                .addGroup(layout.createSequentialGroup()
169.                    .addContainerGap()
170.                    .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING,
    false)
171.                        .addComponent(jLabel6, javax.swing.GroupLayout.Alignment.LEADING, javax.sw
    ing.GroupLayout.DEFAULT_SIZE, 274, Short.MAX_VALUE)
172.                        .addComponent(jLabel7, javax.swing.GroupLayout.Alignment.LEADING, javax.sw
    ing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
173.                        .addGroup(javax.swing.GroupLayout.Alignment.LEADING, layout.createSequenti
    alGroup()
174.                            .addGap(10, 10, 10)
175.                            .addComponent(jLabel11)
176.                            .addGap(28, 28, 28)
177.                            .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment
    .BASELINE)
178.                                .addComponent(jLabel12)
179.                                .addComponent(jTextField1, javax.swing.GroupLayout.PREFERRED_SIZE,
    javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
180.                            .addGap(23, 23, 23)
181.                            .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment
    .BASELINE)

```

```

182.                .addComponent(jLabel15)
183.                .addComponent(jTextField2, javax.swing.GroupLayout.PREFERRED_SIZE,
    javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
184.                .addGap(21, 21, 21)
185.                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment
    .BASELINE)
186.                .addComponent(jLabel13)
187.                .addComponent(jComboBox1, javax.swing.GroupLayout.PREFERRED_SIZE,
    javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
188.                .addGap(18, 18, 18)
189.                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment
    .BASELINE)
190.                .addComponent(jLabel14)
191.                .addComponent(jComboBox2, javax.swing.GroupLayout.PREFERRED_SIZE,
    javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
192.                .addGap(18, 18, 18)
193.                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment
    .BASELINE)
194.                .addComponent(jButton1)
195.                .addComponent(jButton2))))
196.            .addContainerGap(25, Short.MAX_VALUE))
197.        );
198.
199.        pack();
200.    }// </editor-fold>
201.
202.    private void jTextField1ActionPerformed(java.awt.event.ActionEvent evt) {
203.        // TODO add your handling code here:
204.    }
205.
206.    private void jComboBox2ActionPerformed(java.awt.event.ActionEvent evt) {
207.        // TODO add your handling code here:
208.    }
209.
210.    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
211.        String keyDES;
212.        int a;
213.        keyDES = JOptionPane.showInputDialog(null, "Enter a key");
214.        a = keyDES.length();
215.        while(a!=8)
216.        {
217.            keyDES = JOptionPane.showInputDialog(null, "Enter a key");
218.            a = keyDES.length();
219.        }
220.
221.        SecretKeySpec skeySpec = new SecretKeySpec(keyDES.getBytes(), "DES");
222.
223.        System.out.println("You choose DES");
224.        fileInput = jTextField1.getText();
225.        fileOutput = jTextField2.getText();
226.        System.out.println("The input file is: " + fileInput + " the outputfile is: "+fileOutp
    ut);
227.
228.        int chooseMetod = jComboBox1.getSelectedIndex();
229.        int chooseAlgorithm = jComboBox2.getSelectedIndex();
230.        try
231.        {
232.            if(chooseMetod == 0)
233.            {
234.                System.out.println("You choose ECB");
235.                try
236.                {

```

```

237.         if(chooseAlgorithm == 0)
238.         {
239.             System.out.println("You choose crypt");
240.             enc = Cipher.getInstance("DES/ECB/NoPadding");
241.             enc.init(Cipher.ENCRYPT_MODE, keySpec);
242.             cifrar(enc);
243.         }
244.         else
245.         {
246.             System.out.println("You choose decrypt");
247.             dec = Cipher.getInstance("DES/ECB/NoPadding");
248.             dec.init(Cipher.DECRYPT_MODE, keySpec);
249.             descifrar(dec);
250.         }
251.     }
252.     catch(Exception e)
253.     {
254.     }
255. }
256. else if(chooseMetod==1)
257. {
258.     System.out.println("You choose CBC");
259.     try
260.     {
261.         if(chooseAlgorithm == 0)
262.         {
263.             System.out.println("You choose crypt");
264.             enc = Cipher.getInstance("DES/CBC/NoPadding");
265.             enc.init(Cipher.ENCRYPT_MODE, keySpec, algorithmParameterSpec);
266.             cifrar(enc);
267.         }
268.         else
269.         {
270.             System.out.println("You choose decrypt");
271.             dec = Cipher.getInstance("DES/CBC/NoPadding");
272.             dec.init(Cipher.DECRYPT_MODE, keySpec, algorithmParameterSpec);
273.             descifrar(dec);
274.         }
275.     }
276.     catch(Exception e)
277.     {
278.     }
279. }
280. else if(chooseMetod == 2)
281. {
282.     System.out.println("You choose CFB");
283.     try
284.     {
285.         if(chooseAlgorithm == 0)
286.         {
287.             System.out.println("You choose crypt");
288.             enc = Cipher.getInstance("DES/CFB/NoPadding");
289.             enc.init(Cipher.ENCRYPT_MODE, keySpec, algorithmParameterSpec);
290.             cifrar(enc);
291.         }
292.         else
293.         {
294.             System.out.println("You choose decrypt");
295.             dec = Cipher.getInstance("DES/CFB/NoPadding");
296.             dec.init(Cipher.DECRYPT_MODE, keySpec, algorithmParameterSpec);
297.             cifrar(dec);
298.         }
299.     }
300.     catch(Exception e)
301.     {

```

```

302.         }
303.     }
304.     else if(chooseMetod == 3)
305.     {
306.         System.out.println("You choose OFB");
307.         try
308.         {
309.             if(chooseAlgorithm == 0)
310.             {
311.                 System.out.println("You choose crypt");
312.                 enc = Cipher.getInstance("DES/OFB/NoPadding");
313.                 enc.init(Cipher.ENCRYPT_MODE, keySpec, algorithmParameterSpec);
314.                 cifrar(enc);
315.             }
316.             else
317.             {
318.                 System.out.println("You choose decrypt");
319.                 dec = Cipher.getInstance("DES/OFB/NoPadding");
320.                 dec.init(Cipher.DECRYPT_MODE, keySpec, algorithmParameterSpec);
321.                 cifrar(dec);
322.             }
323.         }
324.         catch(Exception e)
325.         {
326.         }
327.     }
328. }
329. catch(Exception e)
330. {
331.     e.printStackTrace();
332. }
333.
334. }
335.
336.
337. private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
338.
339.     String keyAES;
340.     int a;
341.     keyAES = JOptionPane.showInputDialog(null, "Enter a key");
342.     a = keyAES.length();
343.     while(a!=16)
344.     {
345.         keyAES = JOptionPane.showInputDialog(null, "Enter a key");
346.         a = keyAES.length();
347.     }
348.     SecretKeySpec keySpec = new SecretKeySpec(keyAES.getBytes(), "AES");
349.     System.out.println("You choose AES");
350.     fileInput = jTextField1.getText();
351.     fileOutput = jTextField2.getText();
352.     System.out.println("The input file is: " + fileInput + " the outputfile is: "+fileOutp
ut);
353.
354.     int chooseMetod = jComboBox1.getSelectedIndex();
355.     int chooseAlgorithm = jComboBox2.getSelectedIndex();
356.     try
357.     {
358.         if(chooseMetod == 0)
359.         {
360.             System.out.println("You choose ECB");
361.             try
362.             {
363.                 if(chooseAlgorithm == 0)
364.                 {

```

```

365.         System.out.println("You choose crypt");
366.         enc = Cipher.getInstance("AES/ECB/NoPadding");
367.         enc.init(Cipher.ENCRYPT_MODE, skeySpec);
368.         cifrar(enc);
369.     }
370.     else
371.     {
372.         System.out.println("You choose decrypt");
373.         dec = Cipher.getInstance("AES/ECB/NoPadding");
374.         dec.init(Cipher.DECRYPT_MODE, skeySpec);
375.         descifrar(dec);
376.     }
377. }
378. catch(Exception e)
379. {
380. }
381. }
382. else if(chooseMetod==1)
383. {
384.     System.out.println("You choose CBC");
385.     try
386.     {
387.         if(chooseAlgorithm == 0)
388.         {
389.             System.out.println("You choose crypt");
390.             enc = Cipher.getInstance("AES/CBC/NoPadding");
391.             enc.init(Cipher.ENCRYPT_MODE, skeySpec, algorithmParameterSpecAES);
392.             cifrar(enc);
393.         }
394.         else
395.         {
396.             System.out.println("You choose decrypt");
397.             dec = Cipher.getInstance("AES/CBC/NoPadding");
398.             dec.init(Cipher.DECRYPT_MODE, skeySpec, algorithmParameterSpecAES);
399.             descifrar(dec);
400.         }
401.     }
402.     catch(Exception e)
403.     {
404.     }
405. }
406. else if(chooseMetod == 2)
407. {
408.     System.out.println("You choose CFB");
409.     try
410.     {
411.         if(chooseAlgorithm == 0)
412.         {
413.             System.out.println("You choose crypt");
414.             enc = Cipher.getInstance("AES/CFB/NoPadding");
415.             enc.init(Cipher.ENCRYPT_MODE, skeySpec, algorithmParameterSpecAES);
416.             cifrar(enc);
417.         }
418.         else
419.         {
420.             System.out.println("You choose decrypt");
421.             dec = Cipher.getInstance("AES/CFB/NoPadding");
422.             dec.init(Cipher.DECRYPT_MODE, skeySpec, algorithmParameterSpecAES);
423.             cifrar(dec);
424.         }
425.     }
426.     catch(Exception e)
427.     {
428.     }
429. }

```

```

430.         else if(chooseMetod == 3)
431.         {
432.             System.out.println("You choose OFB");
433.             try
434.             {
435.                 if(chooseAlgorithm == 0)
436.                 {
437.                     System.out.println("You choose crypt");
438.                     enc = Cipher.getInstance("AES/OFB8/NoPadding");
439.                     enc.init(Cipher.ENCRYPT_MODE, skeySpec, algorithmParameterSpecAES);
440.                     cifrar(enc);
441.                 }
442.                 else
443.                 {
444.                     System.out.println("You choose decrypt");
445.                     dec = Cipher.getInstance("AES/OFB8/NoPadding");
446.                     dec.init(Cipher.DECRYPT_MODE, skeySpec, algorithmParameterSpecAES);
447.                     cifrar(dec);
448.                 }
449.             }
450.             catch(Exception e)
451.             {
452.             }
453.         }
454.     }
455.     catch(Exception e)
456.     {
457.         e.printStackTrace();
458.     }
459. }
460.
461. private void cifrar(Cipher encr) throws IOException, IllegalBlockSizeException, BadPadding
Exception
462. {
463.     BufferedImage img = ImageIO.read(new File(fileInput));
464.     ImageIcon imagen = new ImageIcon(img);
465.     ImageIcon icono = new ImageIcon(imagen.getImage().getScaledInstance(jLabel6.getWidth()
, jLabel6.getHeight(), Image.SCALE_DEFAULT));
466.     jLabel6.setIcon(icono);
467.     this.repaint();
468.     ByteArrayOutputStream baos = new ByteArrayOutputStream();
469.     ImageIO.write(img, "bmp", baos);
470.     baos.flush();
471.     byte[] imgByte = baos.toByteArray();
472.     baos.close();
473.     byte[] totalBytes = new byte[(int)imgByte.length];
474.     byte[] encrypBytes = new byte[(int)imgByte.length-headerZ];
475.     System.out.println("Total length of image: " + (int)imgByte.length);
476.     System.out.println("Length to cipher: " + ((int)imgByte.length-headerZ));
477.
478.     for(int x=0, y=0; x< imgByte.length; x++ )
479.     {
480.         if(x<headerZ)
481.         {
482.             totalBytes[x] = imgByte[x];
483.         }
484.         else
485.         {
486.             encrypBytes[y++] = imgByte[x];
487.         }
488.     }
489.
490.     byte[] newImage = encr.doFinal(encrypBytes);
491.
492.     for(int i = headerZ, j=0; i<imgByte.length;i++,j++ )

```



```

493.         {
494.             totalBytes[i] = newImage[j];
495.         }
496.
497.         InputStream in = new ByteArrayInputStream(totalBytes);
498.         BufferedImage bImageFromConvert = ImageIO.read(in);
499.         ImageIO.write(bImageFromConvert, "bmp", new File(fileOutput));
500.         BufferedImage imgn = ImageIO.read(new File(fileOutput));
501.         ImageIcon imag = new ImageIcon(imgn);
502.         ImageIcon ic = new ImageIcon(imag.getImage().getScaledInstance(jLabel7.getWidth(), jLa
bel7.getHeight(), Image.SCALE_DEFAULT));
503.         jLabel7.setIcon(ic);
504.         this.repaint();
505.     }
506.
507.     private void descifrar(Cipher desc) throws IOException, IllegalBlockSizeException, BadPadd
ingException
508.     {
509.         BufferedImage img2 = ImageIO.read(new File(fileInput));
510.         ImageIcon imagen = new ImageIcon(img2);
511.         ImageIcon icono = new ImageIcon(imagen.getImage().getScaledInstance(jLabel6.getWidth()
, jLabel6.getHeight(), Image.SCALE_DEFAULT));
512.         jLabel6.setIcon(icono);
513.         this.repaint();
514.         ByteArrayOutputStream b02 = new ByteArrayOutputStream();
515.         ImageIO.write(img2, "bmp", b02);
516.         b02.flush();
517.         byte[] imgByte2 = b02.toByteArray();
518.         b02.close();
519.
520.         byte[] totalBytes2 = new byte[(int)imgByte2.length];
521.         byte[] encrypBytes2 = new byte[(int)imgByte2.length-headerZ];
522.
523.         System.out.println("Total length of image: " + (int)imgByte2.length);
524.         System.out.println("Length to cipher: " + ((int)imgByte2.length-headerZ));
525.
526.         for(int x=0, y=0; x< imgByte2.length; x++ )
527.         {
528.             if(x<headerZ)
529.             {
530.                 totalBytes2[x] = imgByte2[x];
531.             }
532.             else
533.             {
534.                 encrypBytes2[y++] = imgByte2[x];
535.             }
536.         }
537.
538.         byte[] newImage2 = desc.doFinal(encrypBytes2);
539.
540.         for(int i = headerZ, j=0; i<imgByte2.length;i++,j++ )
541.         {
542.             totalBytes2[i] = newImage2[j];
543.         }
544.         InputStream in2 = new ByteArrayInputStream(totalBytes2);
545.         BufferedImage bImageFromConvert2 = ImageIO.read(in2);
546.         ImageIO.write(bImageFromConvert2, "bmp", new File(fileOutput));
547.         BufferedImage imgn2 = ImageIO.read(new File(fileOutput));
548.         ImageIcon imag2 = new ImageIcon(imgn2);
549.         ImageIcon ic2 = new ImageIcon(imag2.getImage().getScaledInstance(jLabel7.getWidth(), j
Label7.getHeight(), Image.SCALE_DEFAULT));
550.         jLabel7.setIcon(ic2);
551.         this.repaint();
552.     }
553.     public static void main(String args[])

```

```

554.     {
555.         java.awt.EventQueue.invokeLater(new Runnable()
556.         {
557.             public void run()
558.             {
559.                 try {
560.                     new desaes().setVisible(true);
561.                 } catch (NoSuchAlgorithmException ex) {
562.                     Logger.getLogger(desaes.class.getName()).log(Level.SEVERE, null, ex);
563.                 }
564.             }
565.         });
566.     }
567.
568.     // Variables declaration - do not modify
569.     private javax.swing.JButton jButton1;
570.     private javax.swing.JButton jButton2;
571.     private javax.swing.JComboBox jComboBox1;
572.     private javax.swing.JComboBox jComboBox2;
573.     private javax.swing.JLabel jLabel1;
574.     private javax.swing.JLabel jLabel2;
575.     private javax.swing.JLabel jLabel3;
576.     private javax.swing.JLabel jLabel4;
577.     private javax.swing.JLabel jLabel5;
578.     private javax.swing.JLabel jLabel6;
579.     private javax.swing.JLabel jLabel7;
580.     private javax.swing.JTextField jTextField1;
581.     private javax.swing.JTextField jTextField2;
582.     // End of variables declaration
583. }

```