INSTITUTO POLITÉCNICO NACIONAL
ESCUELA SUPERIOR DE CÓMPUTO

Cryptography

"Operation modes"

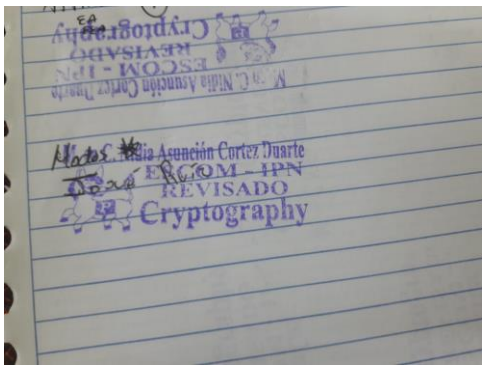
Abstact

Here we are going to see how to encrypt and decrypt using 4 types of operation modes, and how to apply them to encrypt images.

By:

Josué Ruíz Hernández

Professor:
MSc. NIDIA ASUNCIÓN CORTEZ DUARTE



March 2017

Contenido

Introduction

Block cipher is an encryption algorithm which takes fixed size of input say *b* bits and produces a ciphertext of *b* bits again. If input is larger than *b* bits it can be divided further. For different applications and uses, there are several modes of operations for a block cipher.

There exist 4 modes of operation: Electronic Cipher Block, Cipher Block Channing, Cipher Feed Back, Output Feed Back and all have both advantages and disadvantages, that's why one mode is more useful than other one in one situation, for example ECB is not very convenient if we are going to cipher images with few colors like the Japanese flag.

These operations modes can cipher plain texts, images and whatever we want if they can be separated in blocks. The block size depends of the algorithm that we are going to use, for example the block size is of 64 bytes if we are going to use the DES algorithm, but if we want to use the AES algorithm the block size must be of 128 bytes.

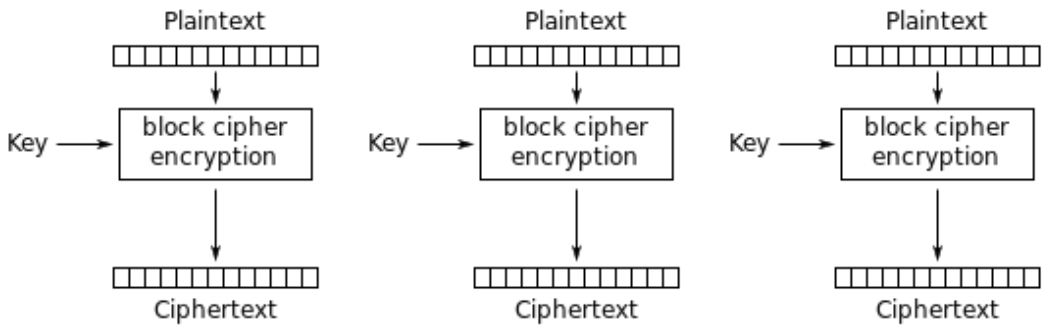
We can use others cypher algorithms like Hill but is not as efficient like AES one, but we can implement manually easier than if we want to implement DES algorithm without using the libraries.

We can cipher using this operation modes in a lot of programming languages like python, rubi, c++ and java. For this experiment we are going to use java as our programming language because it is easier to use in terms of interfaces and it count with a lot of methods that can make us the work easier.

Literature review

Electronic Code Book (ECB)

Electronic code book is the easiest block cipher mode of functioning. It is easier because of direct encryption of each block of input plaintext and output is in form of blocks of encrypted ciphertext. Generally, if a message is larger than *b* bits in size, it can be broken down into bunch of blocks and the procedure is repeated.



Electronic Codebook (ECB) mode encryption

The encryption formula is:

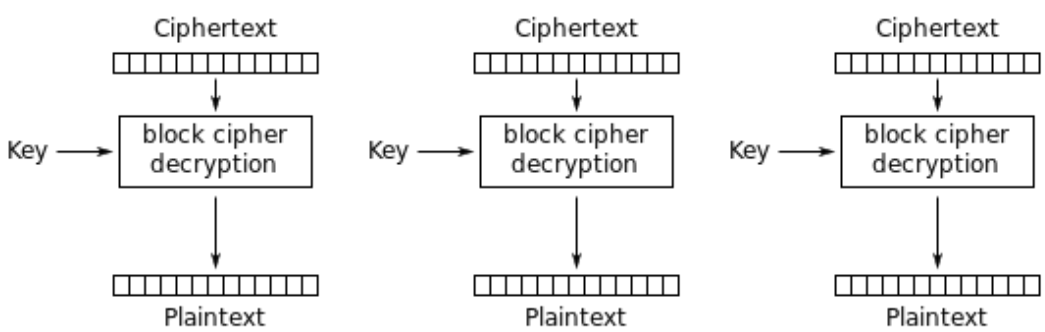
$$C_j = E(P_j)$$

Where:

C is the block j cyphered of the plain text

E is the cypher function (AES/DES/HILL)

P is the block j of the plain text



Electronic Codebook (ECB) mode decryption

The decryption formula is:

$$P_j = D(C_j)$$

Where

C is the block j cyphered of the plain text

D is the decipher function (AES/DES/HILL)

P is the block j of the plain text

Advantages of using ECB –

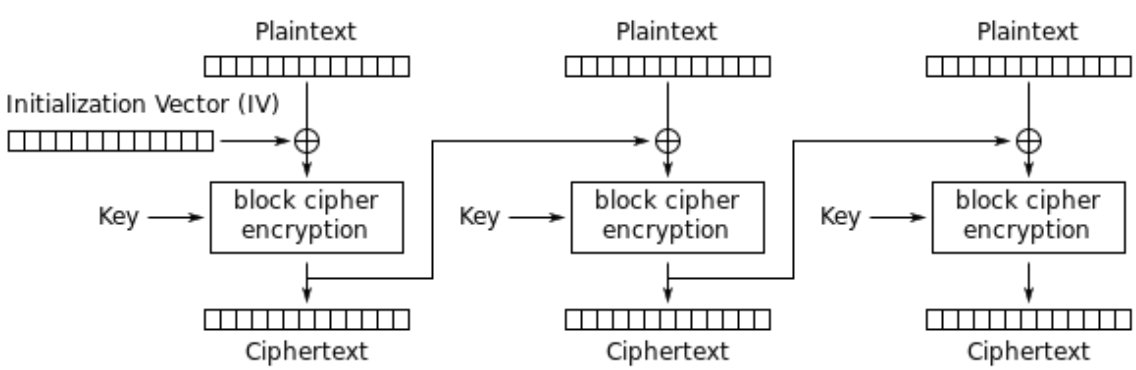
- Parallel encryption of blocks of bits is possible, thus it is a faster way of encryption.
- Simple way of block cipher.

Disadvantages of using ECB –

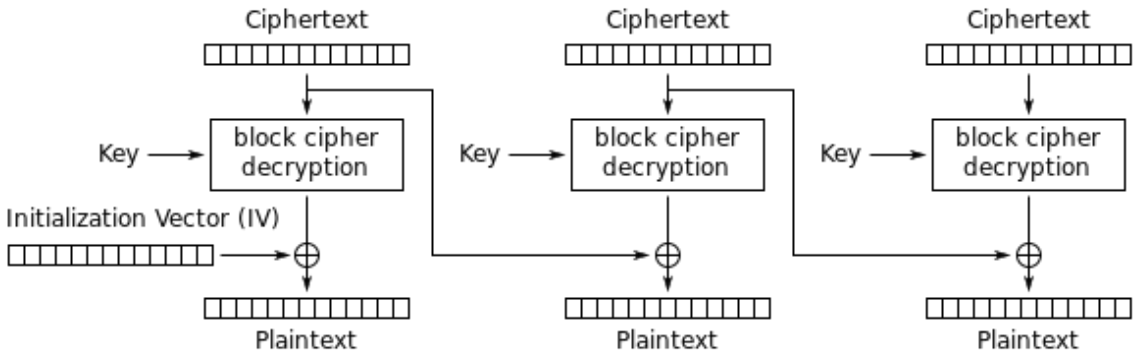
- Prone to cryptanalysis since there is a direct relationship between plaintext and ciphertext.

Cipher Block Chaining

Cipher block chaining or CBC is an advancement made on ECB since ECB compromises some security requirements. In CBC, previous cipher block is given as input to next encryption algorithm after XOR with original plaintext block. In a nutshell here, a cipher block is produced by encrypting a XOR output of previous cipher block and present plaintext block.



Cipher Block Chaining (CBC) mode encryption



Cipher Block Chaining (CBC) mode decryption

The encryption equation is:

$$C_j = E(P_j \oplus C_{j-1})$$

The decryption equation is:

$$P_j = D(C_j) \oplus C_{j-1}$$

Advantages of CBC

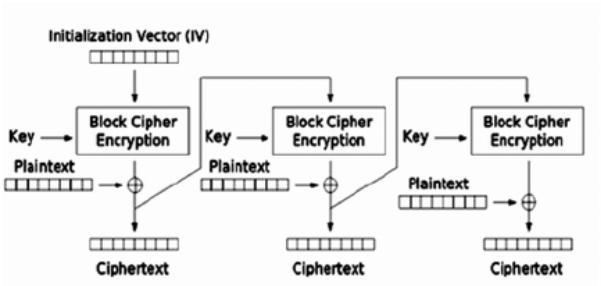
- CBC works well for input greater than *b* bits.
- CBC is a good authentication mechanism.
- Better resistive nature towards cryptanalsis than ECB.

Disadvantages of CBC

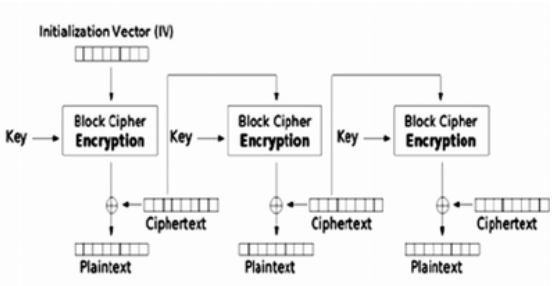
- Parallel encryption is not possible since every encryption requires previous cipher.

Cipher Feedback Mode (CFB)

In this mode the cipher is given as feedback to the next block of encryption with some new specifications: first an initial vector IV is used for first encryption and output bits are divided as set of *s* and *b-s* bits the left-hand side *s*-bits are selected and are applied an XOR operation with plaintext bits. The result given as input to a shift register and the process continues. The encryption and decryption process for the same is shown below, both use encryption algorithm.



Cipher Feedback (CFB) mode encryption



Cipher Feedback (CFB) mode decryption

The encryption equation is:

$$C_j = E(C_{j-1}) \oplus P_j$$

The decryption equation is:

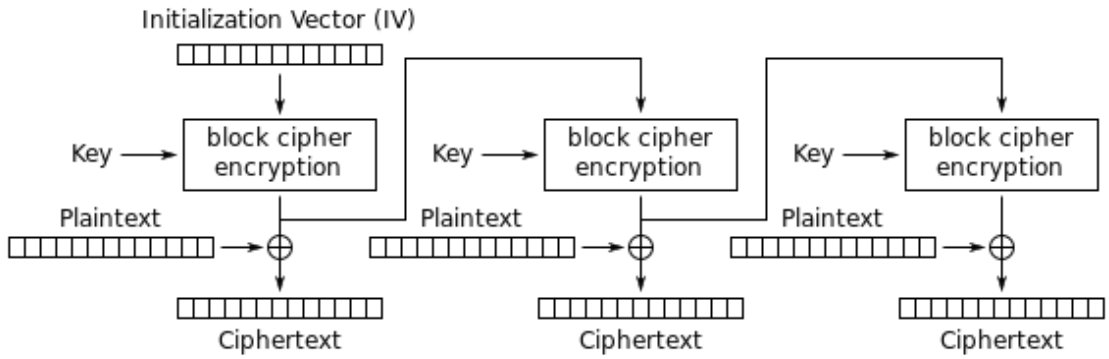
$$P_j = C_j \oplus D(C_{j-1})$$

Advantages of CFB

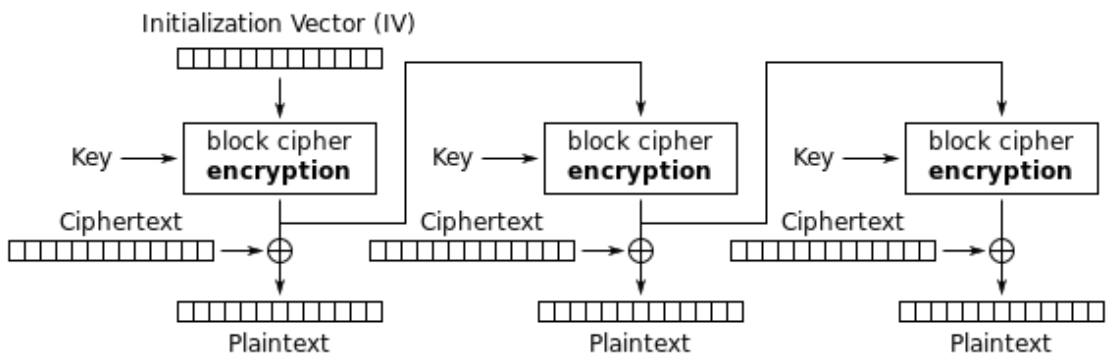
- Since, there is some data loss due to use of shift register, thus it is difficult for applying cryptanalysis.

Output Feedback Mode

The output feedback mode follows nearly same process as the Cipher Feedback mode except that it sends the encrypted output as feedback instead of the actual cipher which is XOR output. In this output feedback mode, all bits of the block are sent instead of sending selected *s* bits. The Output Feedback mode of block cipher holds great resistance towards bit transmission errors. It also decreases dependency or relationship of cipher on plaintext.



Output Feedback (OFB) mode encryption



Output Feedback (OFB) mode decryption

The encryption equation is:

$$C_j = E_j(C_0) \oplus P_j$$

Where the j is the number of the block and E_j the times that C_0 has been encrypted.

The decryption equation is:

$$P_j = D_j(C_0) \oplus C_j$$

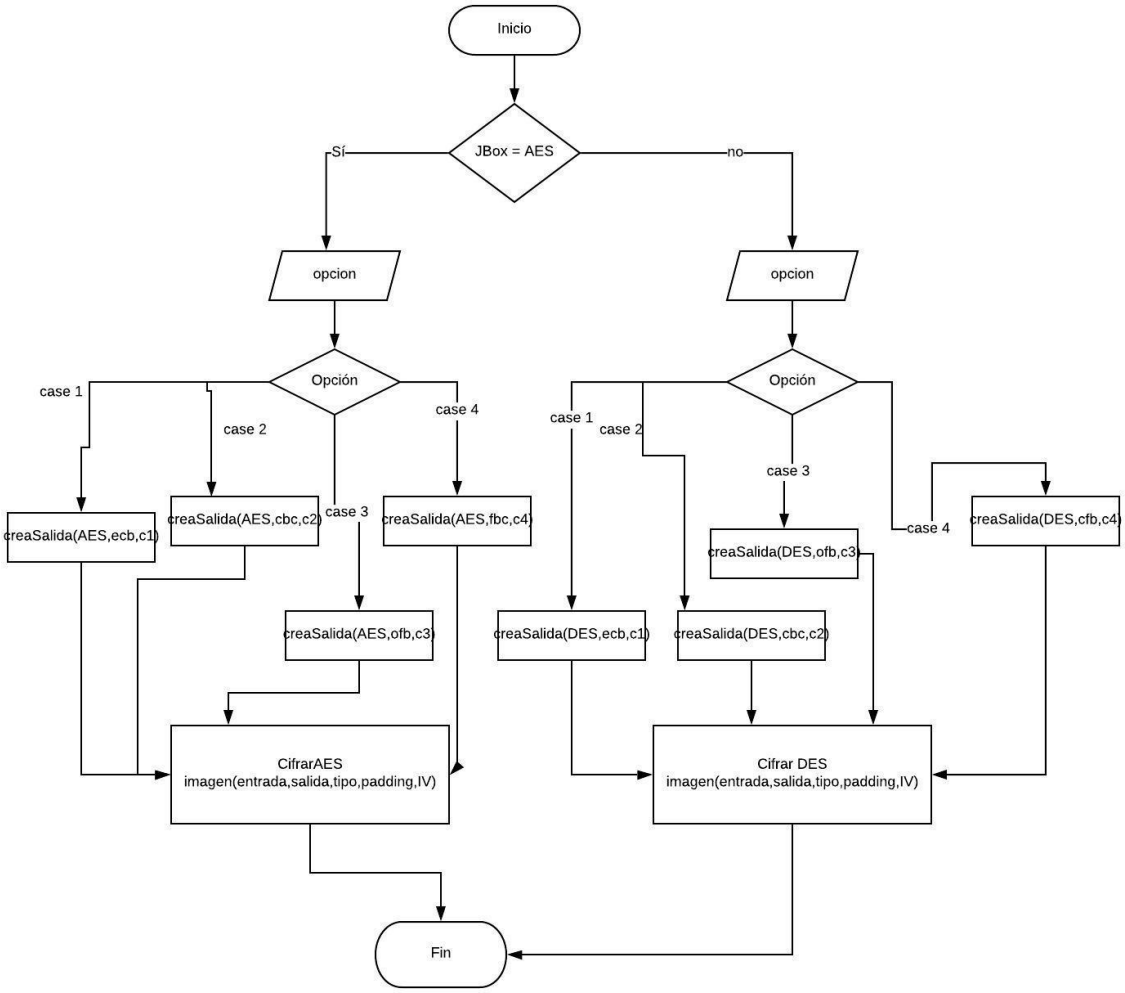
Where the j is the number of the block and $D_j$is the times that C_0 has been decrypted.

Software

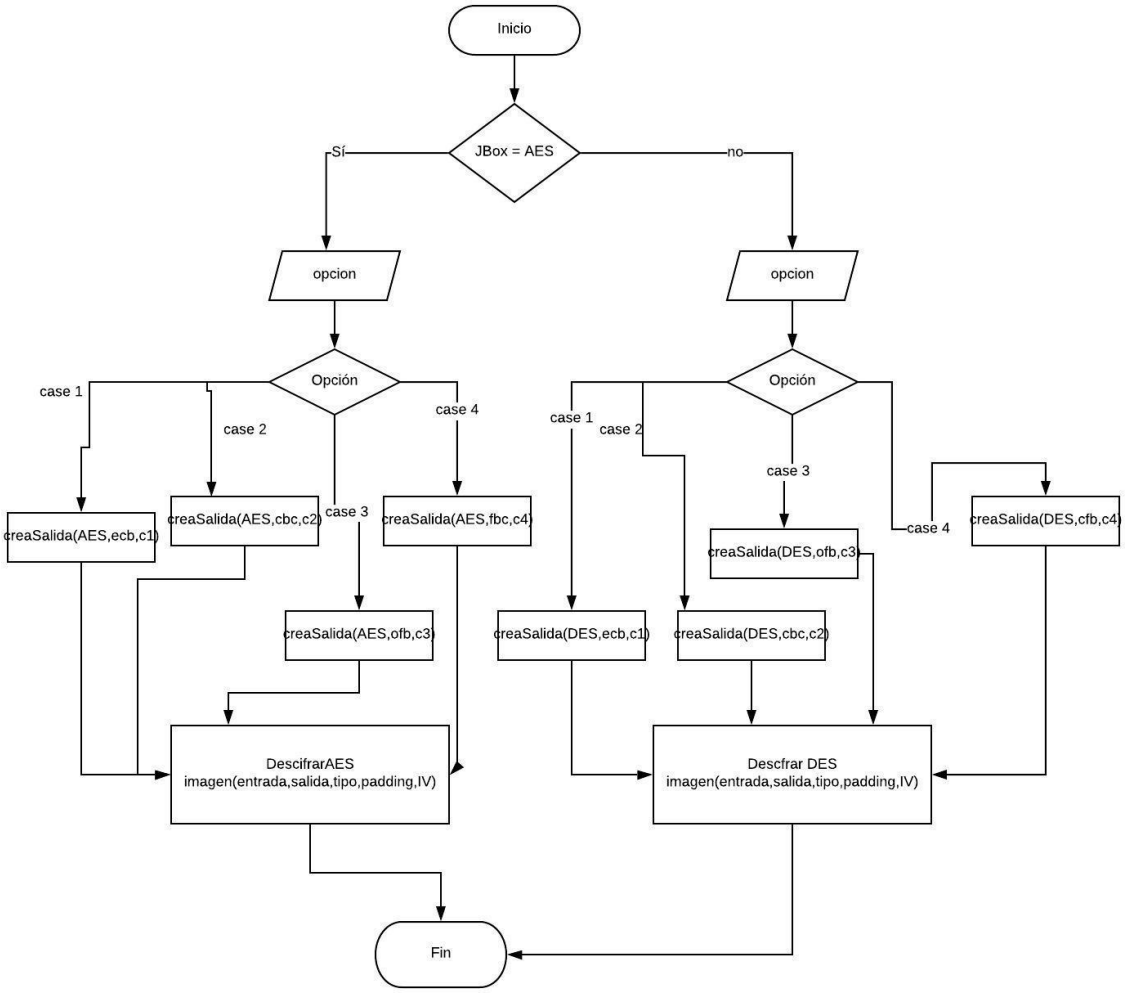
1. NetBeans as the development environment
2. Java as the programming language
3. Javax.crypto.* as the library to encrypt and decrypt the images

Procedure

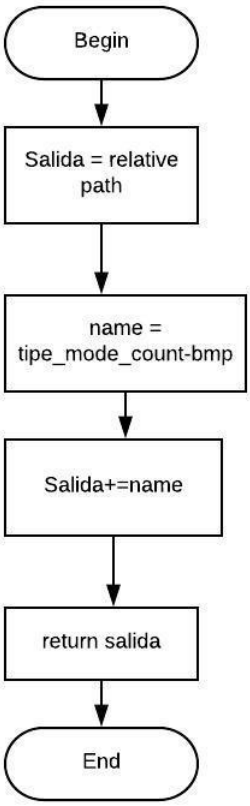
Encrypt



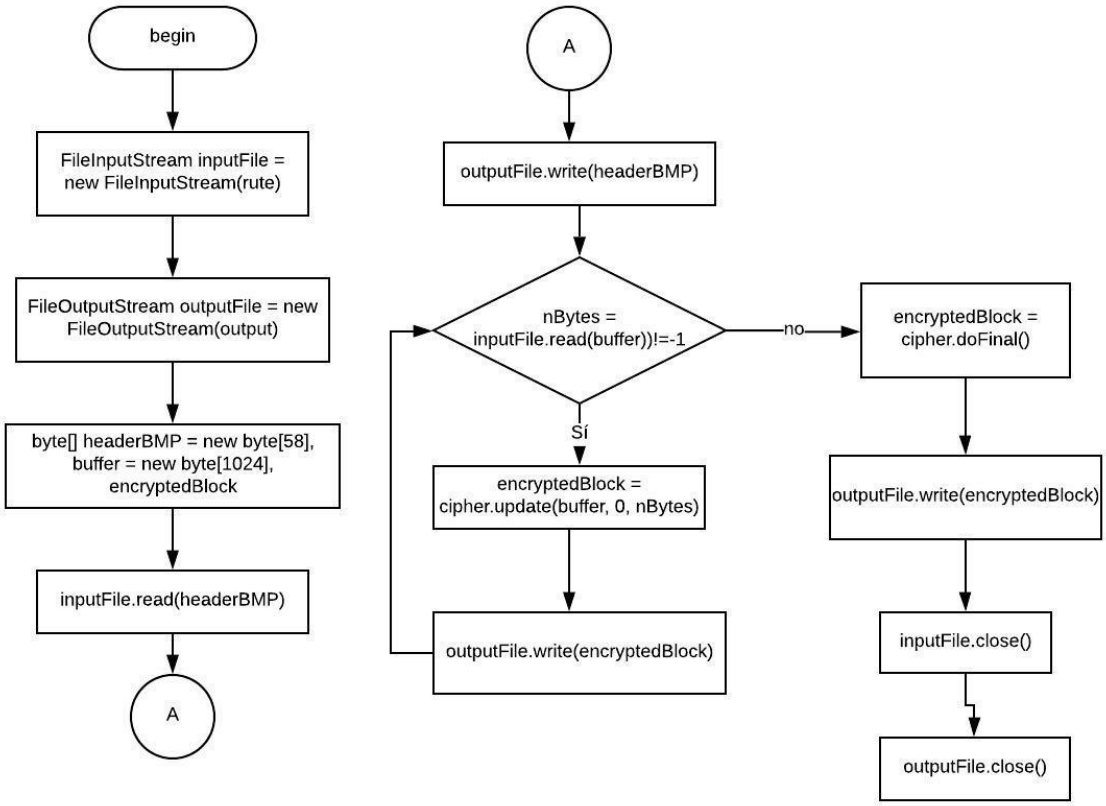
Decrypt


Inicio
JBox = AES
Sí
no
opcion
opcion
Opción
Opción
case 1
case 2
case 3
case 4
case 1
case 2
case 3
case 4
creaSalida(AES,ecb,c1)
creaSalida(AES,cbc,c2)
creaSalida(AES,fbc,c4)
creaSalida(AES,ofb,c3)
creaSalida(DES,ecb,c1)
creaSalida(DES,cbc,c2)
creaSalida(DES,ofb,c3)
creaSalida(DES,cfb,c4)
DescifrarAES
imagen(entrada,salida,tipo,padding,IV)
Descfrar DES
imagen(entrada,salida,tipo,padding,IV)
Fin

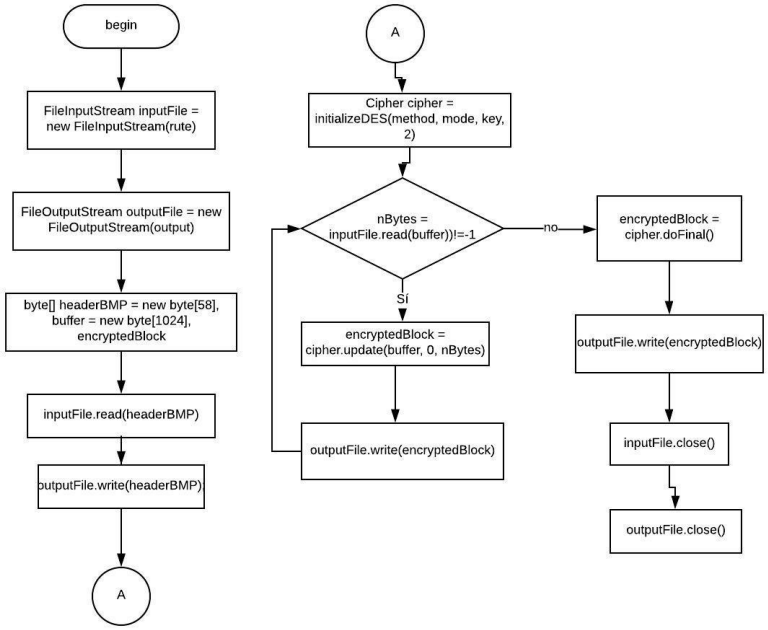
Create output



Encrypt



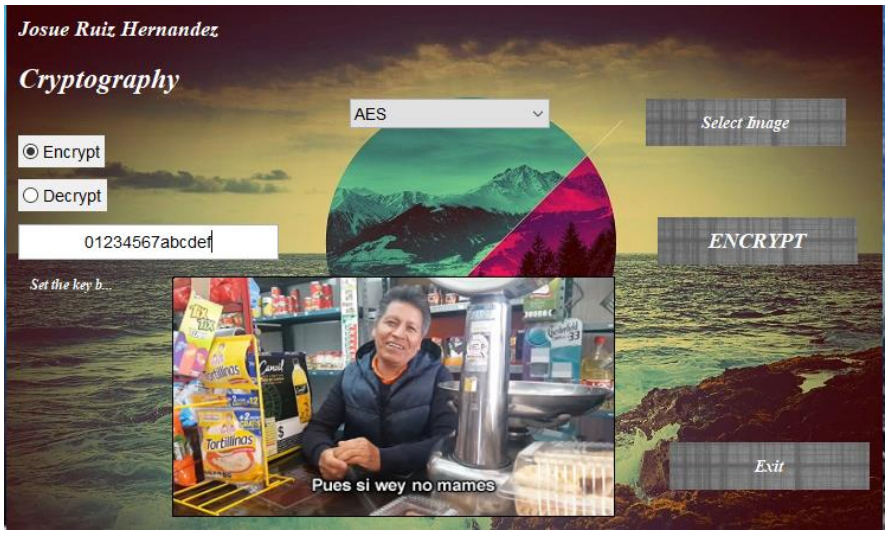
Decrypt



Results

Let’s make a proof using the meme:

For this example, we are going to use the AES algorithm

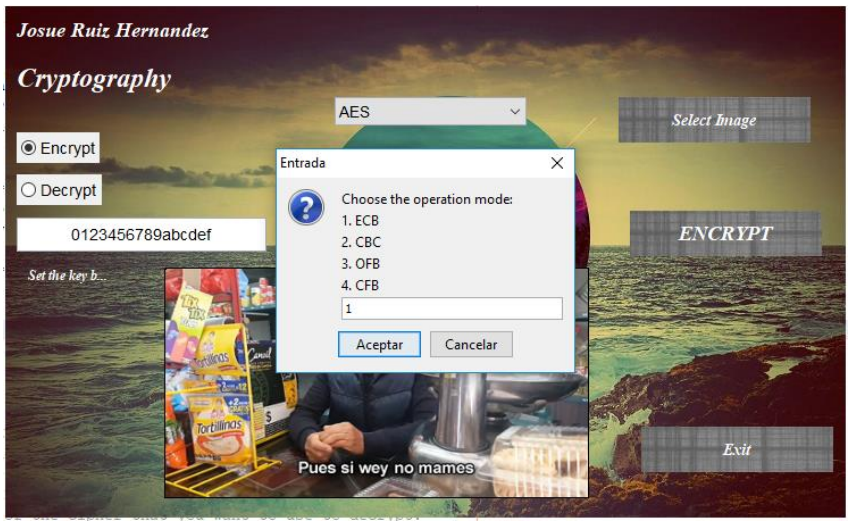


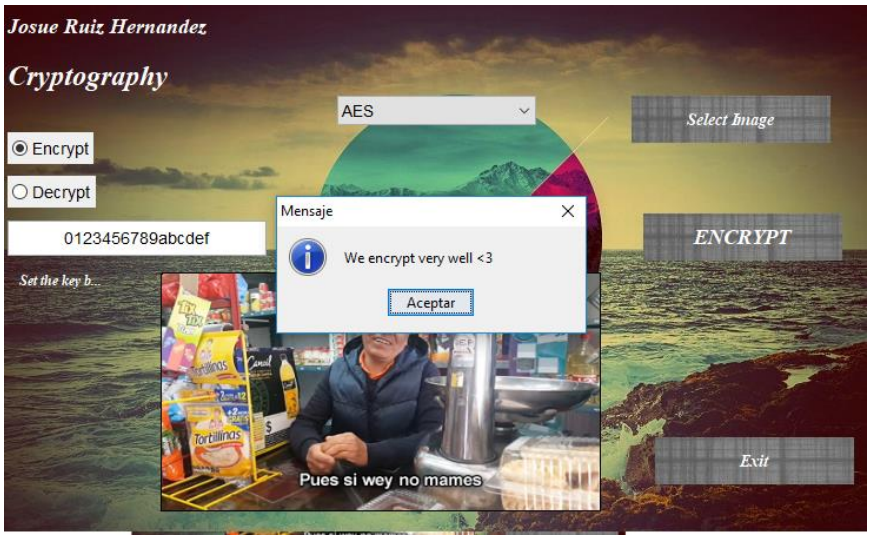
We are going to encrypt it in the order:

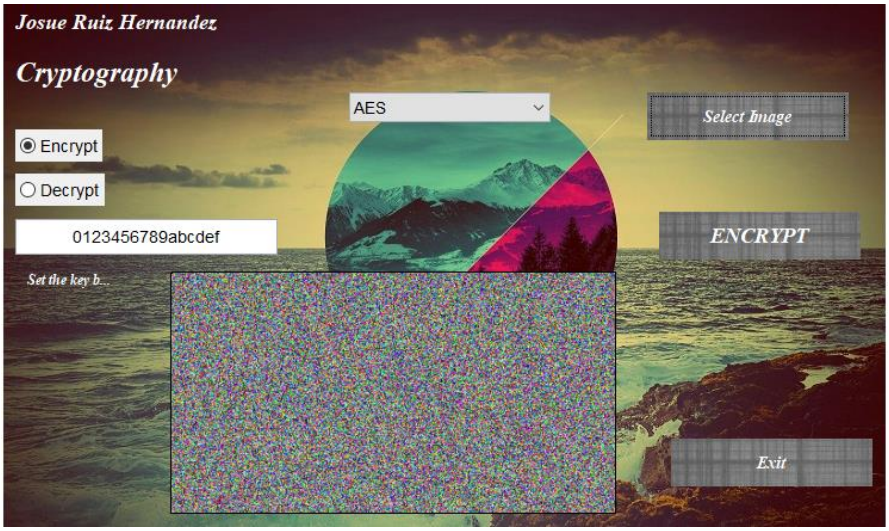
1. ECB
2. CBC
3. OFB
4. CFB

And obviously the order to decrypt it is the same

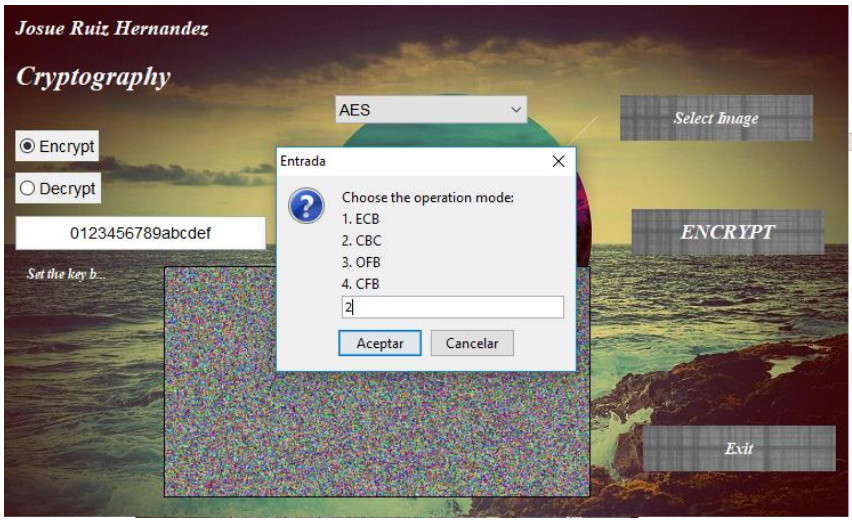
1. Encrypt with ECB:

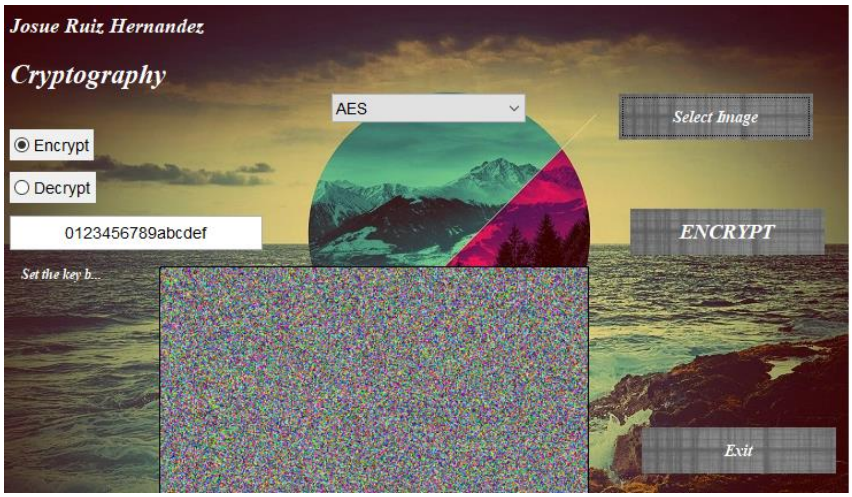




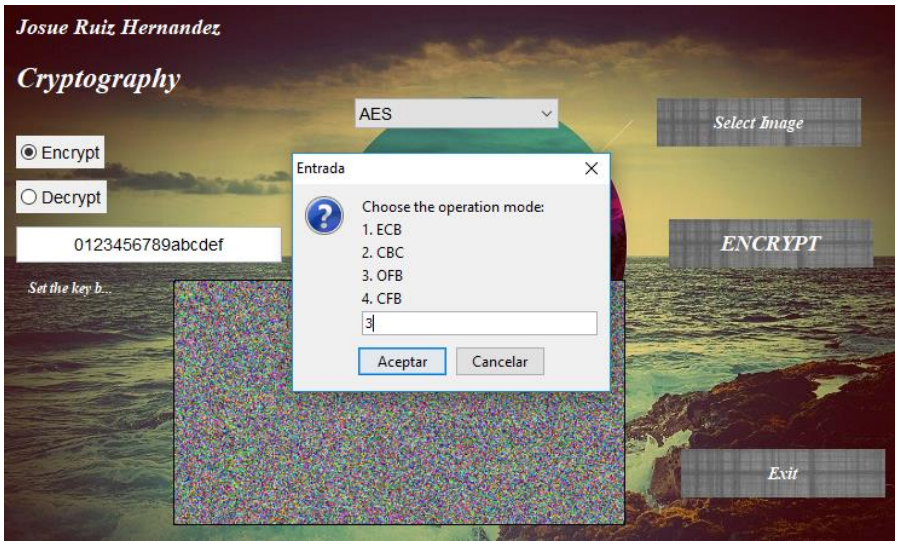


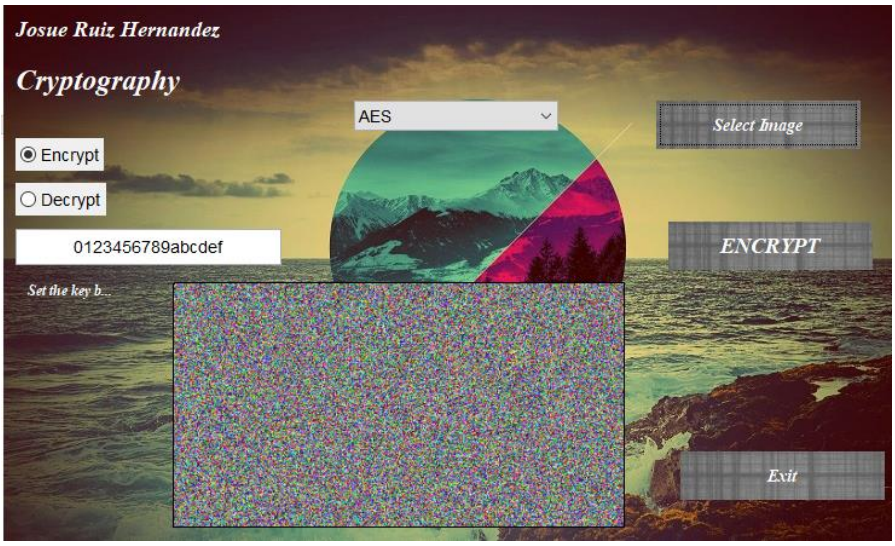
2. Encrypt it using CBC



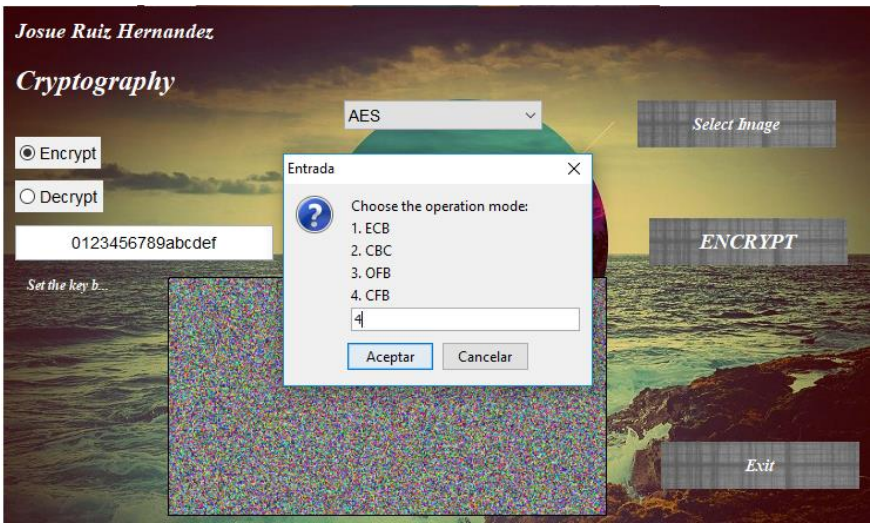


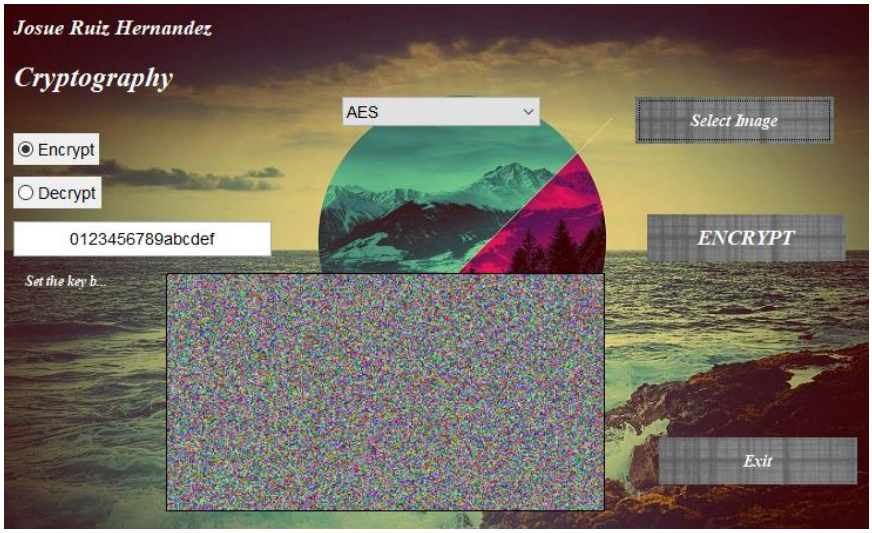
3. Using OFB


Josue Ruiz Hernandez
Cryptography
AES
Select Image
Encrypt
Decrypt
0123456789abcdef
Set the key b...
Entrada
Choose the operation mode:
1. ECB
2. CBC
3. OFB
4. CFB
3
Aceptar
Cancelar
ENCRYPT
Exit

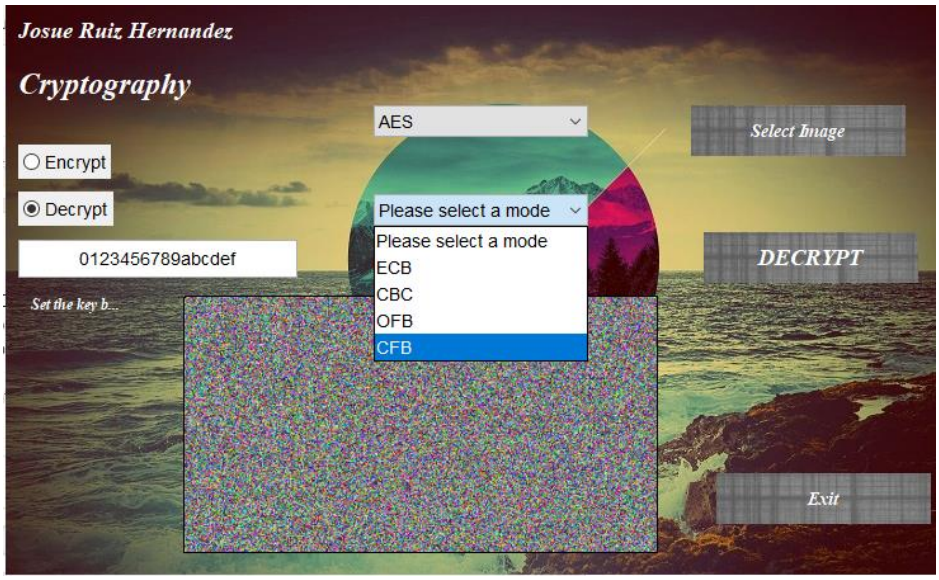

Josue Ruiz Hernandez
Cryptography
AES
Select Image
Encrypt
Decrypt
0123456789abcdef
Set the key b...
ENCRYPT
Exit

4. Using CFB

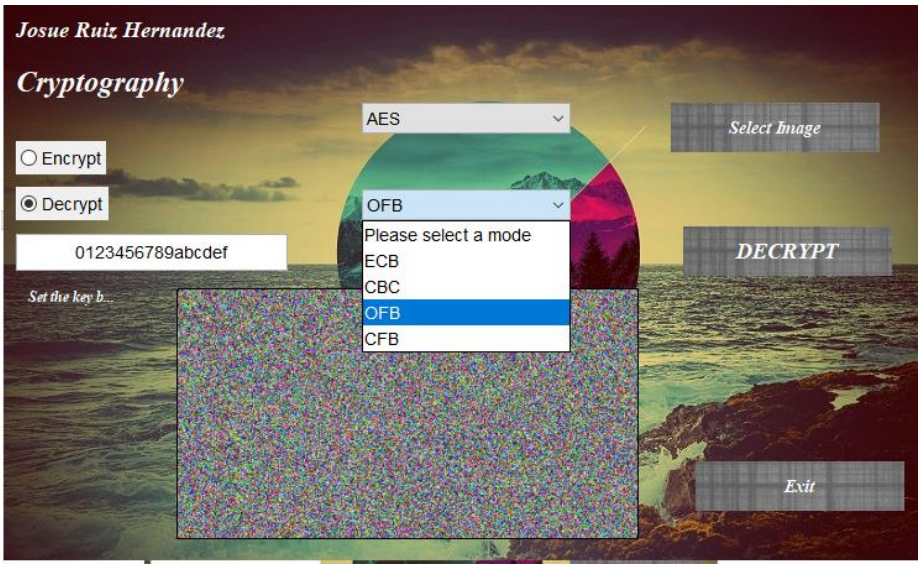


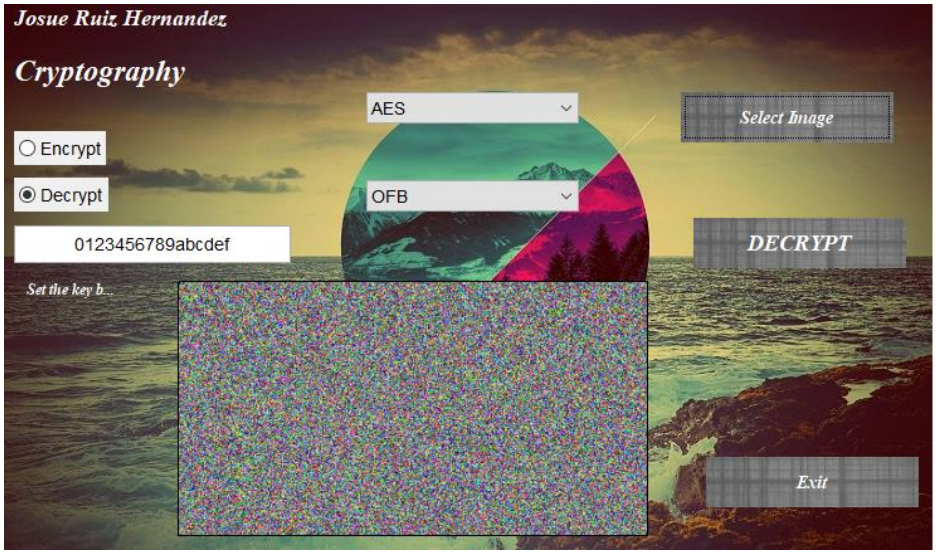


5. Let´s decrypt using CFB

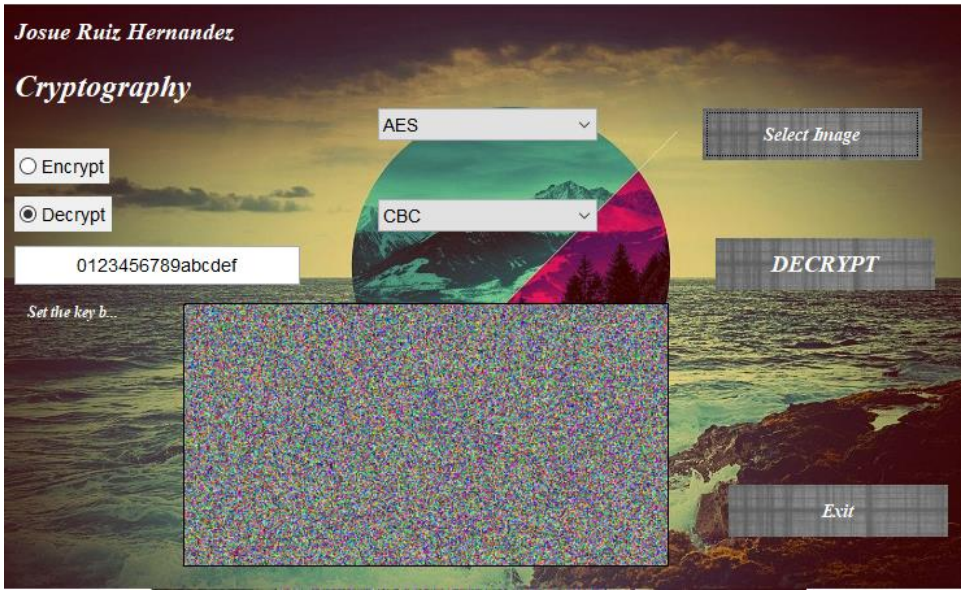


6. Now using OFB

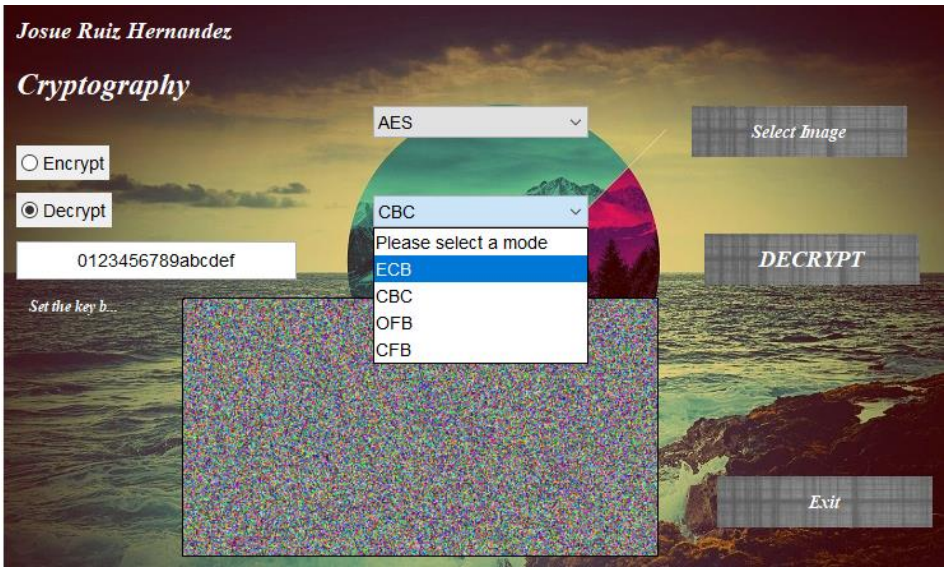

Josue Ruiz Hernandez
Cryptography
AES
Select Image
Encrypt
Decrypt
OFB
Please select a mode
ECB
CBC
OFB
CFB
0123456789abcdef
DECRYPT
Set the key b...
Exit

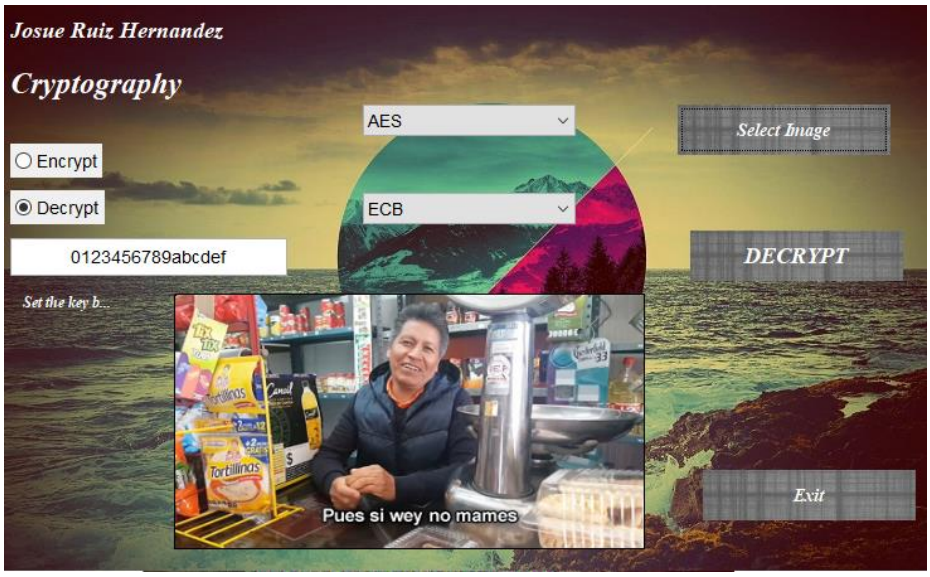

Josue Ruiz Hernandez
Cryptography
AES
Select Image
Encrypt
Decrypt
OFB
0123456789abcdef
DECRYPT
Set the key b...
Exit

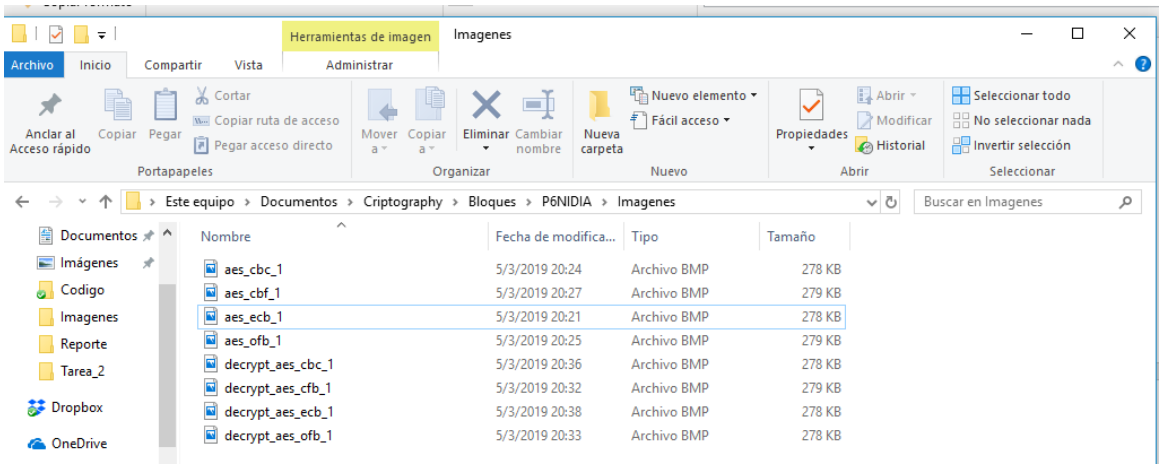
7. Using CBC



8. Finally using EBC


Josue Ruiz Hernandez
Cryptography
AES
Select Image
Encrypt
Decrypt
CBC
Please select a mode
ECB
CBC
OFB
CFB
0123456789abcdef
DECRYPT
Set the key b...
Exit

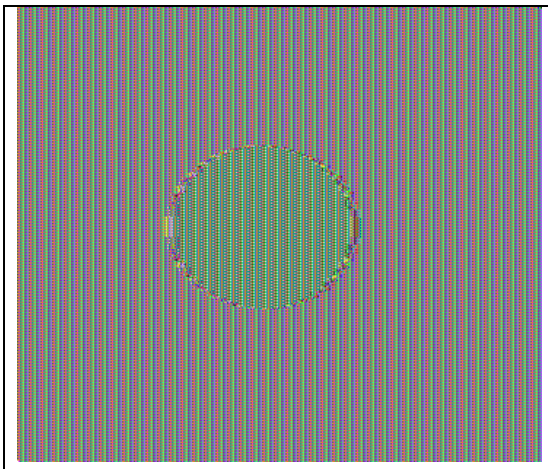

Josue Ruiz Hernandez
Cryptography
AES
Select Image
Encrypt
Decrypt
ECB
0123456789abcdef
DECRYPT
Set the key b...
Pues si wey no mames
Exit

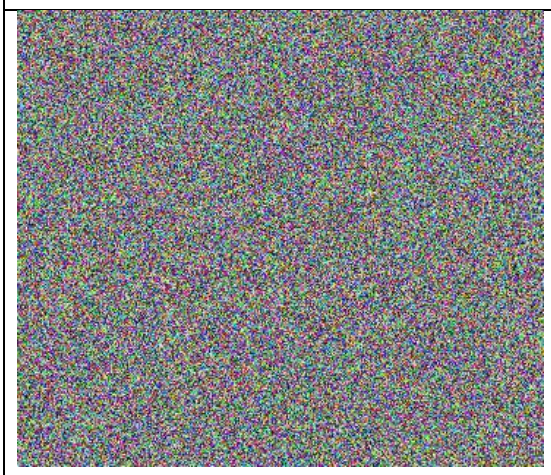


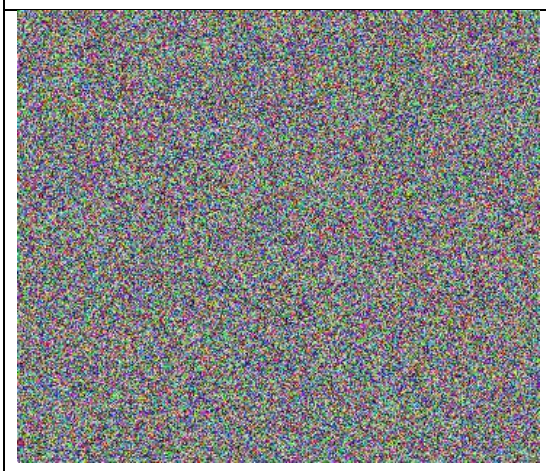
That's are the images that the program created

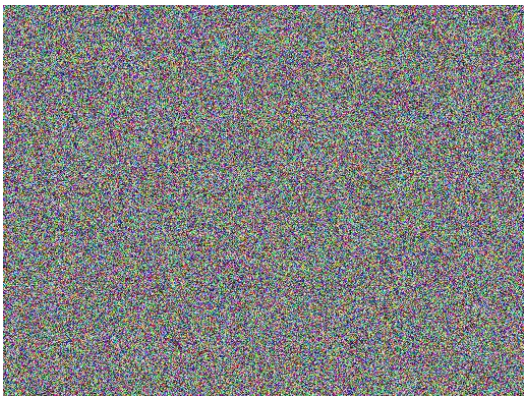
I made the proof with 2 images, the first one is the Japanese flag and the second one is a landscape, the results are showed in the next table:

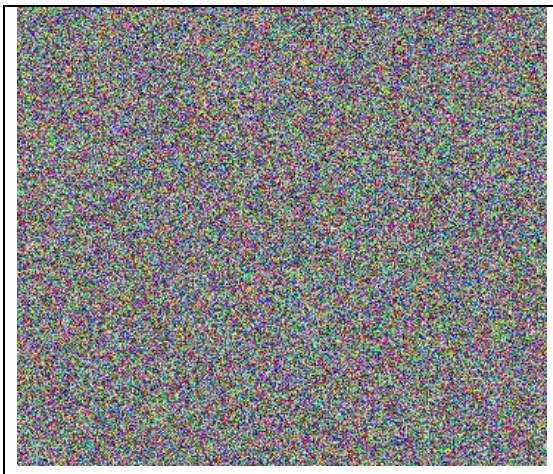
Japanese flag with AES	Landscape with AES
	
	

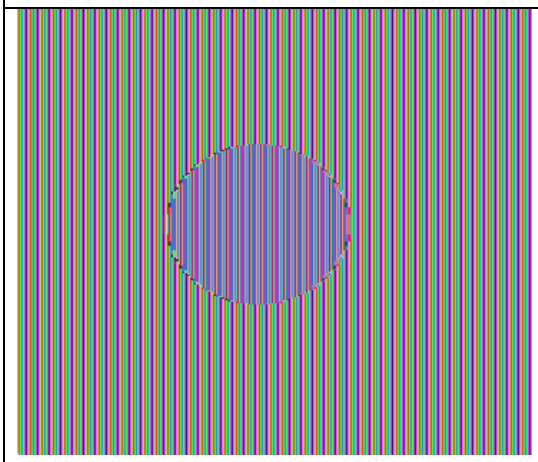


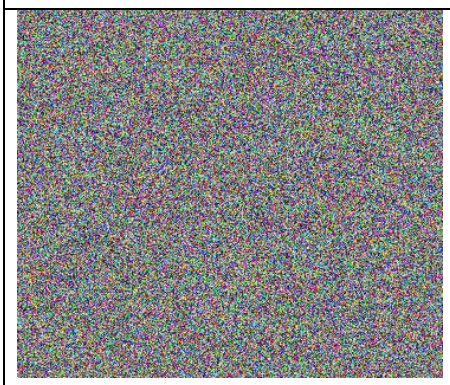
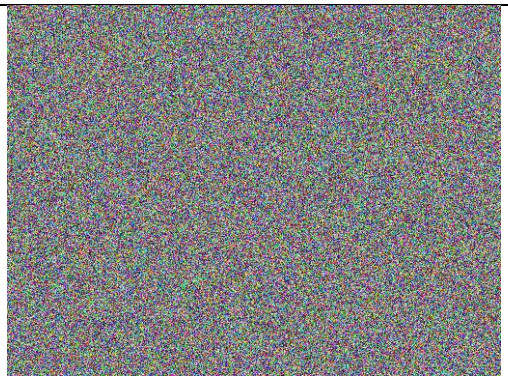






Japanese flag with DES	Landscape with DES
	
	

Discussions

This practice was one of the hardest practices that I have done, first at all I didn't know hot to implement the operation modes in Java, before I used the AES cypher to encrypt a text and decrypt it again, that's why I knew more less how to implement the algorithm in this practice but I didn't know how to separate the bytes in blocks to cypher them and use the padding, the fiery otters and me tried in a lot of ways to finally realize that we can cypher the entire block and get it cyphered using the correct padding in the function´s parameter. I though that we had to create a lot of functions, one per mode and encryption algorithm, 16 functions to encrypt and 16 to decrypt but instead of that only made 4, 2 to encrypt and 2 to decrypt only changing the parameters in the A/D button using cases. If we decide to cypher the image then instead of choosing the mode we made all the modes and create an image for those modes, but if the user decide decrypt one image, then he can decide the operation mode in which de want to decrypt it.

It was hard, but we finally could <3

Conclusions

The operation modes let us make block ciphers using cryptography methods like AES, DES or Hill. Depending the operation mode that we choose the cipher would be more or less complex in our text or image, for example in the Japanese flag if we use the ECB mode we can still see the circle in the middle of the image and if one person catch it interfering the communication he can realize that I´m sending an image related to the Japanese flag, but if I send a image with a lot of figures and colors ECB work perfectly. All the modes have advantages and disadvantages, for example in CBC if one block is corrupted then the next block is going to be wrong too.

References

Paar, C. Pelzl ,J. Preneel B. (2009) *"Understanding Cryptography: A textbook for students and practitioners."* United States of America: Ed. Springer Verlag. ISBN-13: 978-3642041006.

Stallings, W. (2010) *"Cryptography and network security."* (5a Ed.). United States of America: Ed. Prentice Hall. ISBN-13: 978-00136097044.

Anonimous. (15-06-2017). *Operation modes*. Geek fors Geeks. Geek Recuperado de https://www.geeksforgeeks.org/computer-network-block-cipher-modes-of-operation/

Code

```java
package pack;

import java.awt.Image;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.security.InvalidAlgorithmParameterException;
import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;
import javax.crypto.Cipher;
import javax.crypto.NoSuchPaddingException;
import javax.crypto.SecretKey;
import javax.crypto.SecretKeyFactory;
import javax.crypto.spec.DESKeySpec;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.SecretKeySpec;
import javax.imageio.ImageIO;
import javax.swing.Icon;
import javax.swing.ImageIcon;
import javax.swing.JFileChooser;
import javax.swing.JLabel;
import javax.swing.filechooser.FileNameExtensionFilter;

/**
 *
 * @author Josue
 This clas contains all the Methods to be used in the Practice of AES and DES ciphers
 */
public class Methods {


    /**
     * Obtain the rute of a file
     * @return Returns a String with the rute of the file that is selected on the FileChooser if the rute can not be obtained returns null
    */
    public String rute(){
        FileNameExtensionFilter filter = new FileNameExtensionFilter("Imagen BMP", "bmp");
        String rute = "";
        File d = new File("C:\\Users\\josue\\Documents\\Criptography\\Bloques\\P6NIDIA\\Imagenes");
        try {
        JFileChooser chooser = new JFileChooser();
        chooser.setFileFilter(filter);
        chooser.setDialogTitle("SELECT BMP");
        chooser.setCurrentDirectory(d);
        int option = chooser.showOpenDialog(chooser);
        if(option == JFileChooser.APPROVE_OPTION){
            File f = chooser.getSelectedFile();
            rute = f.getAbsolutePath();
            return rute;
            }
        }
        catch(Exception e) {
            System.out.println(e.getMessage());
```

```
55.         }
56.         return null;
57.     }
58.
59.     /**
60.      * Set an image in a JLabel
61.      * @param rute The string that has rute of the image
62.      * @param label The JLabel where you want to put the image
63.      * @throws java.io.IOException
64.     */
65.     public void showImg(String rute, JLabel label) throws IOException{
66.         File f = new File(rute);
67.         Image image = ImageIO.read(f);
68.         ImageIcon icon = new ImageIcon(image);
69.         Icon icono = new ImageIcon(icon.getImage().getScaledInstance(label.getWidt
    h(), label.getHeight(), Image.SCALE_SMOOTH));
70.         label.setText(null);
71.         label.setIcon(icono);
72.     }
73.
74.     /**
75.      * Encrypt the image with the AES method
76.      * @param rute The string that has rute of the image
77.      * @param output The string of the path where you want to save the encrypted i
    mage
78.      * @param method The method that you want to use to cipher.
79.      * @param mode The mode of the cipher that you want to use to encrypt
80.      * @param key The key to pass in to the encryption function
81.      * @throws java.lang.Exception
82.     */
83.     public void encrypt(String rute, String output, String method, String mode, St
    ring key) throws Exception {
84.             int nBytes;
85.             FileInputStream inputFile = new FileInputStream(rute);
86.             FileOutputStream outputFile = new FileOutputStream(output);
87.             byte[] headerBMP = new byte[58], buffer = new byte[1024], encryptedBlo
    ck;
88.             inputFile.read(headerBMP);
89.             outputFile.write(headerBMP);
90.             Cipher cipher = initialize(method, mode, key, 1);
91.             while ((nBytes = inputFile.read(buffer))!=-1) {
92.                 encryptedBlock = cipher.update(buffer, 0, nBytes);
93.                 outputFile.write(encryptedBlock);
94.             }
95.             encryptedBlock = cipher.doFinal();
96.             outputFile.write(encryptedBlock);
97.
98.             inputFile.close();
99.             outputFile.close();
100.            }
101.
102.            /**
103.             * Encrypt the image with the AES method
104.             * @param rute The string that has rute of the image
105.             * @param output The string of the path where you want to save the enc
    rypted image
106.             * @param method The method that you want to use to cipher.
107.             * @param mode The mode of the cipher that you want to use to encrypt
108.             * @param key The key to pass in to the encryption function
109.             * @param iv The initializacion vector to encrypt the first block of d
    ata.
```

```
110.             * @throws Exception
111.             */
112.            public void encrypt(String rute, String output, String method, String
    mode, String key, String iv) throws Exception {
113.                    int nBytes;
114.                    FileInputStream inputFile = new FileInputStream(rute);
115.                    FileOutputStream outputFile = new FileOutputStream(output);
116.                    byte[] headerBMP = new byte[58], buffer = new byte[1024], encr
    yptedBlock;
117.                    inputFile.read(headerBMP);
118.                    outputFile.write(headerBMP);
119.                    Cipher cipher = initialize(method, mode, key, iv, 1);
120.                    while ((nBytes = inputFile.read(buffer))!=-1) {
121.                        encryptedBlock = cipher.update(buffer, 0, nBytes);
122.                        outputFile.write(encryptedBlock);
123.                    }
124.                    encryptedBlock = cipher.doFinal();
125.                    outputFile.write(encryptedBlock);
126.
127.                    inputFile.close();
128.                    outputFile.close();
129.            }
130.
131.            /**
132.             * Decrypt the image with the AES method
133.             * @param rute The string that has rute of the encrypted image
134.             * @param output The string of the path where you want to save the dec
    rypted image
135.             * @param method The method that you want to use to decrypt.
136.             * @param mode The mode of the cipher that you want to use to decrypt.

137.             * @param key The key to pass in to the dcryption function
138.             * @throws Exception
139.             */
140.            public void decrypt(String rute, String output, String method, String
    mode, String key) throws Exception {
141.                    int nBytes;
142.                    FileInputStream inputFile = new FileInputStream(rute);
143.                    FileOutputStream outputFile = new FileOutputStream(output);
144.                    byte[] headerBMP = new byte[58], buffer = new byte[1024], encr
    yptedBlock;
145.                    inputFile.read(headerBMP);
146.                    outputFile.write(headerBMP);
147.                    Cipher cipher = initialize(method, mode, key, 2);
148.                    while ((nBytes = inputFile.read(buffer))!=-1) {
149.                        encryptedBlock = cipher.update(buffer, 0, nBytes);
150.                        outputFile.write(encryptedBlock);
151.                    }
152.                    encryptedBlock = cipher.doFinal();
153.                    outputFile.write(encryptedBlock);
154.
155.                    inputFile.close();
156.                    outputFile.close();
157.            }
158.
159.            /**
160.             * Decrypt the image with the AES method
161.             * @param rute The string that has rute of the encrypted image
162.             * @param output The string of the path where you want to save the dec
    rypted image
163.             * @param method The method that you want to use to decrypt.
```

```
164.             * @param mode The mode of the cipher that you want to use to decrypt.
165.             * @param key The key to pass in to the dcryption function
166.             * @param iv The initializacion vector to decrypt the first block of d
    ata.
167.             * @throws Exception
168.             */
169.            public void decrypt(String rute, String output, String method, String
    mode, String key, String iv) throws Exception {
170.                    int nBytes;
171.                    FileInputStream inputFile = new FileInputStream(rute);
172.                    FileOutputStream outputFile = new FileOutputStream(output);
173.                    byte[] headerBMP = new byte[58], buffer = new byte[1024], encr
    yptedBlock;
174.                    inputFile.read(headerBMP);
175.                    outputFile.write(headerBMP);
176.                    Cipher cipher = initialize(method, mode, key, iv, 2);
177.                    while ((nBytes = inputFile.read(buffer))!=-1) {
178.                        encryptedBlock = cipher.update(buffer, 0, nBytes);
179.                        outputFile.write(encryptedBlock);
180.                    }
181.                    encryptedBlock = cipher.doFinal();
182.                    outputFile.write(encryptedBlock);
183.
184.                    inputFile.close();
185.                    outputFile.close();
186.            }
187.
188.            /**
189.             * Initialize the cipher to the method and the mode that is provide
190.             * @param method The method that is being used to encrypt
191.             * @param mode The mode that is being used to encrypt
192.             * @param key The key to pass in to the decryption function
193.             * @param ed 1 to encript and 2 to decrypt
194.             * @return The cipher to encrypt the image
195.             * @throws NoSuchAlgorithmException
196.             * @throws NoSuchPaddingException
197.             * @throws InvalidKeyException
198.             */
199.            public Cipher initialize(String method, String mode, String key, int e
    d) throws NoSuchAlgorithmException, NoSuchPaddingException, InvalidKeyException{
200.                Cipher cipher = Cipher.getInstance(mode);
201.                SecretKeySpec skeySpec = new SecretKeySpec(key.getBytes(), method)
    ;
202.                switch (ed) {
203.                    case 1:
204.                        cipher.init(Cipher.ENCRYPT_MODE, skeySpec);
205.                        return cipher;
206.                    case 2:
207.                        cipher.init(Cipher.DECRYPT_MODE, skeySpec);
208.                        return cipher;
209.                    default:
210.                        return null;
211.                }
212.            }
213.
214.            /**
215.             * Initialize the cipher to the method and the mode that is provide
216.             * @param method The method that is being used to encrypt
217.             * @param mode The mode that is being used to encrypt
218.             * @param key The key to pass in to the decryption function
```

```
219.             * @param iv The initialization vector
220.             * @param ed 1 to encript and 2 to decrypt
221.             * @return The cipher to encrypt the image
222.             * @throws NoSuchAlgorithmException
223.             * @throws NoSuchPaddingException
224.             * @throws InvalidKeyException
225.             * @throws InvalidAlgorithmParameterException
226.             */
227.            public Cipher initialize(String method, String mode, String key, Strin
    g iv, int ed) throws NoSuchAlgorithmException, NoSuchPaddingException, InvalidKeyE
    xception, InvalidAlgorithmParameterException{
228.                Cipher cipher = Cipher.getInstance(mode);
229.                SecretKeySpec skeySpec = new SecretKeySpec(key.getBytes(), method)
    ;
230.                IvParameterSpec ivParameterSpec = new IvParameterSpec(iv.getBytes(
    ));
231.                switch (ed) {
232.                    case 1:
233.                        cipher.init(Cipher.ENCRYPT_MODE, skeySpec, ivParameterSpec
    );
234.                        return cipher;
235.                    case 2:
236.                        cipher.init(Cipher.DECRYPT_MODE, skeySpec, ivParameterSpec
    );
237.                        return cipher;
238.                    default:
239.                        return null;
240.                }
241.            }
242.
243.            /**
244.             * Initialize the cipher to the method and the mode that is provide th
    e diference between the other method is the form wich the key is cipher
245.             * @param method The method that is being used to encrypt
246.             * @param mode The mode that is being used to encrypt
247.             * @param key The key to pass in to the decryption function
248.             * @param iv The initialization vector
249.             * @param ed 1 to encript and 2 to decrypt
250.             * @return The cipher to encrypt the image
251.             * @throws NoSuchAlgorithmException
252.             * @throws NoSuchPaddingException
253.             * @throws InvalidKeyException
254.             * @throws InvalidAlgorithmParameterException
255.             * @throws Exception
256.             */
257.            public Cipher initializeDES(String method, String mode, String key, St
    ring iv, int ed) throws NoSuchAlgorithmException, NoSuchPaddingException, InvalidK
    eyException, InvalidAlgorithmParameterException, Exception{
258.                Cipher cipher = Cipher.getInstance(mode);
259.                IvParameterSpec ivParameterSpec = new IvParameterSpec(iv.getBytes(
    ));
260.                SecretKey skeySpec = generateKey(key);
261.                switch (ed) {
262.                    case 1:
263.                        cipher.init(Cipher.ENCRYPT_MODE, skeySpec, ivParameterSpec
    );
264.                        return cipher;
265.                    case 2:
266.                        cipher.init(Cipher.DECRYPT_MODE, skeySpec, ivParameterSpec
    );
267.                        return cipher;
```

```
                    default:
                        return null;
                }
            }

            /**
             * Initialize the cipher to the method and the mode that is provide the diference between the other method is the form wich the key is cipher
             * @param method The method that is being used to encrypt
             * @param mode The mode that is being used to encrypt
             * @param key The key to pass in to the decryption function
             * @param ed 1 to encript and 2 to decrypt
             * @return The cipher to encrypt the image
             * @throws NoSuchAlgorithmException
             * @throws NoSuchPaddingException
             * @throws InvalidKeyException
             * @throws InvalidAlgorithmParameterException
             * @throws Exception
             */
            public Cipher initializeDES(String method, String mode, String key, int ed) throws NoSuchAlgorithmException, NoSuchPaddingException, InvalidKeyException, InvalidAlgorithmParameterException, Exception{
                Cipher cipher = Cipher.getInstance(mode);
                SecretKey skeySpec = generateKey(key);
                switch (ed) {
                    case 1:
                        cipher.init(Cipher.ENCRYPT_MODE, skeySpec);
                        return cipher;
                    case 2:
                        cipher.init(Cipher.DECRYPT_MODE, skeySpec);
                        return cipher;
                    default:
                        return null;
                }
            }

            /**
             * Cipher the key provided
             * @param stringKey The string with the key
             * @return SecretKey a cipher key
             * @throws Exception
             */
            public SecretKey generateKey( String stringKey ) throws Exception{
                SecretKeyFactory skf = SecretKeyFactory.getInstance("DES");
                DESKeySpec kspec = new DESKeySpec(stringKey.getBytes());
                SecretKey ks = skf.generateSecret(kspec);
                return ks;
            }

            /**
             * Encrypt the image with the DES method
             * @param rute The string that has rute of the image
             * @param output The string of the path where you want to save the encrypted image
             * @param method The method that you want to use to cipher.
             * @param mode The mode of the cipher that you want to use to encrypt
             * @param key The key to pass in to the encryption function
             * @throws Exception
             */
            public void encryptDES(String rute, String output, String method, String mode, String key) throws Exception {
```

```
324.                    int nBytes;
325.                    FileInputStream inputFile = new FileInputStream(rute);
326.                    FileOutputStream outputFile = new FileOutputStream(output);
327.                    byte[] headerBMP = new byte[58], buffer = new byte[1024], encr
    yptedBlock;
328.                    inputFile.read(headerBMP);
329.                    outputFile.write(headerBMP);
330.                    Cipher cipher = initializeDES(method, mode, key, 1);
331.                    while ((nBytes = inputFile.read(buffer))!=-1) {
332.                        encryptedBlock = cipher.update(buffer, 0, nBytes);
333.                        outputFile.write(encryptedBlock);
334.                    }
335.                    encryptedBlock = cipher.doFinal();
336.                    outputFile.write(encryptedBlock);
337.
338.                    inputFile.close();
339.                    outputFile.close();
340.            }
341.
342.            /**
343.             * Encrypt the image with the DES method
344.             * @param rute The string that has rute of the image
345.             * @param output The string of the path where you want to save the enc
    rypted image
346.             * @param method The method that you want to use to cipher.
347.             * @param mode The mode of the cipher that you want to use to encrypt
348.             * @param key The key to pass in to the encryption function
349.             * @param iv The initializacion vector to encrypt the first block of d
    ata.
350.             * @throws Exception
351.             */
352.            public void encryptDES(String rute, String output, String method, Stri
    ng mode, String key, String iv) throws Exception {
353.                    int nBytes;
354.                    FileInputStream inputFile = new FileInputStream(rute);
355.                    FileOutputStream outputFile = new FileOutputStream(output);
356.                    byte[] headerBMP = new byte[58], buffer = new byte[1024], encr
    yptedBlock;
357.                    inputFile.read(headerBMP);
358.                    outputFile.write(headerBMP);
359.                    Cipher cipher = initializeDES(method, mode, key, iv, 1);
360.                    while ((nBytes = inputFile.read(buffer))!=-1) {
361.                        encryptedBlock = cipher.update(buffer, 0, nBytes);
362.                        outputFile.write(encryptedBlock);
363.                    }
364.                    encryptedBlock = cipher.doFinal();
365.                    outputFile.write(encryptedBlock);
366.
367.                    inputFile.close();
368.                    outputFile.close();
369.            }
370.
371.            /**
372.             * Decrypt the image with the DES method
373.             * @param rute The string that has rute of the encrypted image
374.             * @param output The string of the path where you want to save the dec
    rypted image
375.             * @param method The method that you want to use to decrypt.
376.             * @param mode The mode of the cipher that you want to use to decrypt.

377.             * @param key The key to pass in to the dcryption function
```

```
378.             * @throws Exception
379.             */
380.            public void decryptDES(String rute, String output, String method, Stri
    ng mode, String key) throws Exception {
381.                    int nBytes;
382.                    FileInputStream inputFile = new FileInputStream(rute);
383.                    FileOutputStream outputFile = new FileOutputStream(output);
384.                    byte[] headerBMP = new byte[58], buffer = new byte[1024], encr
    yptedBlock;
385.                    inputFile.read(headerBMP);
386.                    outputFile.write(headerBMP);
387.                    Cipher cipher = initializeDES(method, mode, key, 2);
388.                    while ((nBytes = inputFile.read(buffer))!=-1) {
389.                        encryptedBlock = cipher.update(buffer, 0, nBytes);
390.                        outputFile.write(encryptedBlock);
391.                    }
392.                    encryptedBlock = cipher.doFinal();
393.                    outputFile.write(encryptedBlock);
394.
395.                    inputFile.close();
396.                    outputFile.close();
397.            }
398.
399.            /**
400.             * Decrypt the image with the DES method
401.             * @param rute The string that has rute of the encrypted image
402.             * @param output The string of the path where you want to save the dec
    rypted image
403.             * @param method The method that you want to use to decrypt.
404.             * @param mode The mode of the cipher that you want to use to decrypt.

405.             * @param key The key to pass in to the dcryption function
406.             * @param iv The initializacion vector to decrypt the first block of d
    ata.
407.             * @throws Exception
408.             */
409.            public void decryptDES(String rute, String output, String method, Stri
    ng mode, String key, String iv) throws Exception {
410.                    int nBytes;
411.                    FileInputStream inputFile = new FileInputStream(rute);
412.                    FileOutputStream outputFile = new FileOutputStream(output);
413.                    byte[] headerBMP = new byte[58], buffer = new byte[1024], encr
    yptedBlock;
414.                    inputFile.read(headerBMP);
415.                    outputFile.write(headerBMP);
416.                    Cipher cipher = initializeDES(method, mode, key, iv, 2);
417.                    while ((nBytes = inputFile.read(buffer))!=-1) {
418.                        encryptedBlock = cipher.update(buffer, 0, nBytes);
419.                        outputFile.write(encryptedBlock);
420.                    }
421.                    encryptedBlock = cipher.doFinal();
422.                    outputFile.write(encryptedBlock);
423.
424.                    inputFile.close();
425.                    outputFile.close();
426.            }
427.        }
428.        package pack;
429.
430.        import java.io.IOException;
431.        import javax.swing.JOptionPane;
```

```
432.
433.        /**
434.         *
435.         * @author Josue
436.         */
437.        public class Interfaces extends javax.swing.JFrame {
438.
439.            Methods met = new Methods();
440.            String rute;
441.
442.            public Interfaces() {
443.                initComponents();
444.                this.setLocationRelativeTo(null);
445.                jButtonSelect.setVisible(false);
446.                jComboBoxMode.setVisible(false);
447.                jComboBoxMethod.setVisible(false);
448.                jButtonED.setVisible(false);
449.                jTextFieldkey.setVisible(false);
450.                jLabelrestriction.setVisible(false);
451.            }
452.
453.            /**
454.             * This method is called from within the constructor to initialize the
    form.
455.             * WARNING: Do NOT modify this code. The content of this method is alw
    ays
456.             * regenerated by the Form Editor.
457.             */
458.            @SuppressWarnings("unchecked")
459.            // <editor-
    fold defaultstate="collapsed" desc="Generated Code">
460.            private void initComponents() {
461.
462.                buttonGroup1 = new javax.swing.ButtonGroup();
463.                jRadioButtonEncrypt = new javax.swing.JRadioButton();
464.                jRadioButtonDecrypt = new javax.swing.JRadioButton();
465.                jButtonSelect = new javax.swing.JButton();
466.                jComboBoxMethod = new javax.swing.JComboBox<String>();
467.                jComboBoxMode = new javax.swing.JComboBox<String>();
468.                jButtonED = new javax.swing.JButton();
469.                jLabelImg = new javax.swing.JLabel();
470.                jTextFieldkey = new javax.swing.JTextField();
471.                jLabel2 = new javax.swing.JLabel();
472.                jLabel3 = new javax.swing.JLabel();
473.                jLabelrestriction = new javax.swing.JLabel();
474.                Fondo = new javax.swing.JLabel();
475.
476.                setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE
    );
477.                getContentPane().setLayout(new org.netbeans.lib.awtextra.AbsoluteL
    ayout());
478.
479.                buttonGroup1.add(jRadioButtonEncrypt);
480.                jRadioButtonEncrypt.setFont(new java.awt.Font("Open Sans", 0, 14))
    ; // NOI18N
481.                jRadioButtonEncrypt.setText("Encrypt");
482.                jRadioButtonEncrypt.addMouseListener(new java.awt.event.MouseAdapt
    er() {
483.                    public void mouseClicked(java.awt.event.MouseEvent evt) {
484.                        jRadioButtonEncryptMouseClicked(evt);
485.                    }
```

```
486.                 });
487.                 jRadioButtonEncrypt.addActionListener(new java.awt.event.ActionLis
    tener() {
488.                     public void actionPerformed(java.awt.event.ActionEvent evt) {

489.                         jRadioButtonEncryptActionPerformed(evt);
490.                     }
491.                 });
492.                 getContentPane().add(jRadioButtonEncrypt, new org.netbeans.lib.awt
    extra.AbsoluteConstraints(10, 111, -1, -1));
493.
494.                 buttonGroup1.add(jRadioButtonDecrypt);
495.                 jRadioButtonDecrypt.setFont(new java.awt.Font("Open Sans", 0, 14))
    ; // NOI18N
496.                 jRadioButtonDecrypt.setText("Decrypt");
497.                 jRadioButtonDecrypt.addMouseListener(new java.awt.event.MouseAdapt
    er() {
498.                     public void mouseClicked(java.awt.event.MouseEvent evt) {
499.                         jRadioButtonDecryptMouseClicked(evt);
500.                     }
501.                 });
502.                 getContentPane().add(jRadioButtonDecrypt, new org.netbeans.lib.awt
    extra.AbsoluteConstraints(10, 148, -1, -1));
503.
504.                 jButtonSelect.setFont(new java.awt.Font("Times New Roman", 3, 14))
    ; // NOI18N
505.                 jButtonSelect.setForeground(new java.awt.Color(255, 255, 255));
506.                 jButtonSelect.setIcon(new javax.swing.ImageIcon(getClass().getReso
    urce("/Images/w2.jpg"))); // NOI18N
507.                 jButtonSelect.setText("Select Image");
508.                 jButtonSelect.setHorizontalTextPosition(javax.swing.SwingConstants
    .CENTER);
509.                 jButtonSelect.addActionListener(new java.awt.event.ActionListener(
    ) {
510.                     public void actionPerformed(java.awt.event.ActionEvent evt) {

511.                         jButtonSelectActionPerformed(evt);
512.                     }
513.                 });
514.                 getContentPane().add(jButtonSelect, new org.netbeans.lib.awtextra.
    AbsoluteConstraints(530, 80, 169, 40));
515.
516.                 jComboBoxMethod.setBackground(new java.awt.Color(51, 255, 255));
517.                 jComboBoxMethod.setFont(new java.awt.Font("Open Sans", 0, 14)); //
     NOI18N
518.                 jComboBoxMethod.setModel(new javax.swing.DefaultComboBoxModel(new
    String[] { "Please select a cipher", "AES", "DES" }));
519.                 jComboBoxMethod.addPropertyChangeListener(new java.beans.PropertyC
    hangeListener() {
520.                     public void propertyChange(java.beans.PropertyChangeEvent evt)
     {
521.                         jComboBoxMethodPropertyChange(evt);
522.                     }
523.                 });
524.                 getContentPane().add(jComboBoxMethod, new org.netbeans.lib.awtextr
    a.AbsoluteConstraints(290, 80, -1, -1));
525.
526.                 jComboBoxMode.setFont(new java.awt.Font("Open Sans", 0, 14)); // N
    OI18N
527.                 jComboBoxMode.setModel(new javax.swing.DefaultComboBoxModel(new St
    ring[] { "Please select a mode", "ECB", "CBC", "OFB", "CFB" }));
```

```
528.                getContentPane().add(jComboBoxMode, new org.netbeans.lib.awtextra.
    AbsoluteConstraints(290, 150, 169, -1));
529.
530.                jButtonED.setFont(new java.awt.Font("Times New Roman", 3, 18)); //
     NOI18N
531.                jButtonED.setForeground(new java.awt.Color(255, 255, 255));
532.                jButtonED.setIcon(new javax.swing.ImageIcon(getClass().getResource
    ("/Images/w2.jpg"))); // NOI18N
533.                jButtonED.setText("E/D");
534.                jButtonED.setHorizontalTextPosition(javax.swing.SwingConstants.CEN
    TER);
535.                jButtonED.addActionListener(new java.awt.event.ActionListener() {

536.                    public void actionPerformed(java.awt.event.ActionEvent evt) {

537.                        jButtonEDActionPerformed(evt);
538.                    }
539.                });
540.                getContentPane().add(jButtonED, new org.netbeans.lib.awtextra.Abso
    luteConstraints(550, 180, 169, 40));
541.
542.                jLabelImg.setHorizontalAlignment(javax.swing.SwingConstants.CENTER
    );
543.                jLabelImg.setBorder(new javax.swing.border.LineBorder(new java.awt
    .Color(0, 0, 0), 1, true));
544.                getContentPane().add(jLabelImg, new org.netbeans.lib.awtextra.Abso
    luteConstraints(190, 230, 374, 203));
545.
546.                jTextFieldkey.setFont(new java.awt.Font("Open Sans", 0, 14)); // N
    OI18N
547.                jTextFieldkey.setHorizontalAlignment(javax.swing.JTextField.CENTER
    );
548.                jTextFieldkey.setText("Insert your key");
549.                jTextFieldkey.setNextFocusableComponent(jButtonED);
550.                jTextFieldkey.addFocusListener(new java.awt.event.FocusAdapter() {

551.                    public void focusGained(java.awt.event.FocusEvent evt) {
552.                        jTextFieldkeyFocusGained(evt);
553.                    }
554.                });
555.                jTextFieldkey.addKeyListener(new java.awt.event.KeyAdapter() {
556.                    public void keyPressed(java.awt.event.KeyEvent evt) {
557.                        jTextFieldkeyKeyPressed(evt);
558.                    }
559.                    public void keyTyped(java.awt.event.KeyEvent evt) {
560.                        jTextFieldkeyKeyTyped(evt);
561.                    }
562.                });
563.                getContentPane().add(jTextFieldkey, new org.netbeans.lib.awtextra.
    AbsoluteConstraints(10, 186, 220, 30));
564.
565.                jLabel2.setFont(new java.awt.Font("Times New Roman", 3, 18)); // N
    OI18N
566.                jLabel2.setForeground(new java.awt.Color(255, 255, 255));
567.                jLabel2.setText("Josue Ruiz Hernandez");
568.                getContentPane().add(jLabel2, new org.netbeans.lib.awtextra.Absolu
    teConstraints(10, 10, 180, -1));
569.
570.                jLabel3.setFont(new java.awt.Font("Times New Roman", 3, 24)); // N
    OI18N
571.                jLabel3.setForeground(new java.awt.Color(255, 255, 255));
```

```
572.                jLabel3.setText("Cryptography");
573.                getContentPane().add(jLabel3, new org.netbeans.lib.awtextra.Absolu
    teConstraints(10, 49, 140, -1));
574.
575.                jLabelrestriction.setFont(new java.awt.Font("Times New Roman", 3,
    12)); // NOI18N
576.                jLabelrestriction.setForeground(new java.awt.Color(255, 255, 255))
    ;
577.                jLabelrestriction.setText("Restriction");
578.                getContentPane().add(jLabelrestriction, new org.netbeans.lib.awtex
    tra.AbsoluteConstraints(20, 230, 70, -1));
579.
580.                Fondo.setIcon(new javax.swing.ImageIcon(getClass().getResource("/I
    mages/w1.jpg"))); // NOI18N
581.                getContentPane().add(Fondo, new org.netbeans.lib.awtextra.Absolute
    Constraints(0, 0, 740, 450));
582.
583.                pack();
584.            }// </editor-fold>
585.
586.            private void jRadioButtonEncryptActionPerformed(java.awt.event.ActionE
    vent evt) {
587.
588.            }
589.
590.            private void jRadioButtonDecryptMouseClicked(java.awt.event.MouseEvent
     evt) {
591.                jButtonSelect.setVisible(true);
592.                jComboBoxMethod.setVisible(true);
593.                jTextFieldkey.setVisible(true);
594.                if (jRadioButtonEncrypt.isSelected()) {
595.                    jButtonED.setVisible(true);
596.                    jButtonED.setText("ENCRYPT");
597.                    jComboBoxMode.setVisible(false);
598.                }else{
599.                    jButtonED.setVisible(true);
600.                    jButtonED.setText("DECRYPT");
601.                    jComboBoxMode.setVisible(true);
602.                }
603.            }
604.
605.            private void jRadioButtonEncryptMouseClicked(java.awt.event.MouseEvent
     evt) {
606.                jButtonSelect.setVisible(true);
607.                jComboBoxMethod.setVisible(true);
608.                jTextFieldkey.setVisible(true);
609.                if (jRadioButtonEncrypt.isSelected()) {
610.                    jButtonED.setVisible(true);
611.                    jButtonED.setText("ENCRYPT");
612.                    jComboBoxMode.setVisible(false);
613.
614.                }else{
615.                    jButtonED.setVisible(true);
616.                    jButtonED.setText("DECRYPT");
617.                    jComboBoxMode.setVisible(true);
618.                }
619.            }
620.
621.            private void jButtonSelectActionPerformed(java.awt.event.ActionEvent e
    vt) {
622.                rute = met.rute();
```

```
623.                try {
624.                    met.showImg(rute, jLabelImg);
625.                } catch (IOException ex) {
626.                    System.out.println(ex.getMessage());
627.                }
628.            }
629.
630.            private void jButtonEDActionPerformed(java.awt.event.ActionEvent evt)
    {
631.                if (jRadioButtonEncrypt.isSelected()) {
632.                    switch (jComboBoxMethod.getSelectedIndex()) {
633.                        case 1:
634.                            try {
635.                                met.encrypt(rute, "C:\\Users\\josue\\Documents\\Cr
    iptography\\Bloques\\P6NIDIA\\Imagenes\\AESimg_ecb.bmp", "AES", "AES/ECB/PKCS5Padd
    ing", jTextFieldkey.getText());
636.                                met.encrypt(rute, "C:\\Users\\josue\\Documents\\Cr
    iptography\\Bloques\\P6NIDIA\\Imagenes\\AESimg_cbc.bmp", "AES", "AES/CBC/PKCS5Padd
    ing", jTextFieldkey.getText(), "0123456789ABCDEF");
637.                                met.encrypt(rute, "C:\\Users\\josue\\Documents\\Cr
    iptography\\Bloques\\P6NIDIA\\Imagenes\\AESimg_ofb.bmp", "AES", "AES/OFB/PKCS5Padd
    ing", jTextFieldkey.getText(), "0123456789ABCDEF");
638.                                met.encrypt(rute, "C:\\Users\\josue\\Documents\\Cr
    iptography\\Bloques\\P6NIDIA\\Imagenes\\AESimg_cfb.bmp", "AES", "AES/CFB/PKCS5Padd
    ing", jTextFieldkey.getText(), "0123456789ABCDEF");
639.                                JOptionPane.showMessageDialog(this, "We encrypt ve
    ry well <3");
640.                            } catch (Exception ex) {
641.                                System.out.println(ex.getMessage());
642.                            }   break;
643.                        case 2:
644.                            try {
645.                                met.encryptDES(rute, "C:\\Users\\josue\\Documents\
    \Criptography\\Bloques\\P6NIDIA\\Imagenes\\DESimg_ecb.bmp", "DES", "DES/ECB/PKCS5P
    adding", jTextFieldkey.getText());
646.                                met.encryptDES(rute, "C:\\Users\\josue\\Documents\
    \Criptography\\Bloques\\P6NIDIA\\Imagenes\\DESimg_cbc.bmp", "DES", "DES/CBC/PKCS5P
    adding", jTextFieldkey.getText(), "01234567");
647.                                met.encryptDES(rute, "C:\\Users\\josue\\Documents\
    \Criptography\\Bloques\\P6NIDIA\\Imagenes\\DESimg_ofb.bmp", "DES", "DES/OFB/PKCS5P
    adding", jTextFieldkey.getText(), "01234567");
648.                                met.encryptDES(rute, "C:\\Users\\josue\\Documents\
    \Criptography\\Bloques\\P6NIDIA\\Imagenes\\DESimg_cfb.bmp", "DES", "DES/CFB/PKCS5P
    adding", jTextFieldkey.getText(), "01234567");
649.                                JOptionPane.showMessageDialog(this, "We encrypt ve
    ry well <3");
650.                            } catch (Exception ex) {
651.                                System.out.println(ex.getMessage());
652.                            }   break;
653.                        default:
654.                            JOptionPane.showMessageDialog(this, "SELECT MODE", "OO
    OOOOOOOOPS", JOptionPane.ERROR_MESSAGE);
655.                            break;
656.                    }
657.                }else{
658.                    switch (jComboBoxMethod.getSelectedIndex()) {
659.                        case 1:
660.                            switch (jComboBoxMode.getSelectedIndex()) {
661.                                case 1:
662.                                    try {
```

```
                                        met.decrypt(rute, "C:\\Users\\josue\\Documents\\Criptography\\Bloques\\P6NIDIA\\Imagenes\\decrypt_img_ecb.bmp", "AES", "AES/ECB/PKCS5Padding", jTextFieldkey.getText());
                                        JOptionPane.showMessageDialog(this, "We decrypt very well <3");
                                    } catch (Exception ex) {
                                        System.err.println(ex.toString());
                                        JOptionPane.showMessageDialog(this, "Something wrong dude </3");
                                    }
                                    break;
                                case 2:
                                    try {
                                        met.decrypt(rute, "C:\\Users\\josue\\Documents\\Criptography\\Bloques\\P6NIDIA\\Imagenes\\decrypt_img_cbc.bmp", "AES", "AES/CBC/PKCS5Padding", jTextFieldkey.getText(), "0123456789ABCDEF");
                                        JOptionPane.showMessageDialog(this, "We decrypt very well <3");
                                    } catch (Exception ex) {
                                        System.err.println(ex.toString());
                                        JOptionPane.showMessageDialog(this, "Something wrong dude </3");
                                    }
                                    break;
                                case 3:
                                    try {
                                        met.decrypt(rute, "C:\\Users\\josue\\Documents\\Criptography\\Bloques\\P6NIDIA\\Imagenes\\decrypt_img_ofb.bmp", "AES", "AES/OFB/PKCS5Padding", jTextFieldkey.getText(), "0123456789ABCDEF");
                                        JOptionPane.showMessageDialog(this, "We decrypt very well <3");
                                    } catch (Exception ex) {
                                        System.err.println(ex.toString());
                                        JOptionPane.showMessageDialog(this, "Something wrong dude </3");
                                    }
                                    break;
                                case 4:
                                    try {
                                        met.decrypt(rute, "C:\\Users\\josue\\Documents\\Criptography\\Bloques\\P6NIDIA\\Imagenes\\decrypt_img_cfb.bmp", "AES", "AES/CFB/PKCS5Padding", jTextFieldkey.getText(), "0123456789ABCDEF");
                                        JOptionPane.showMessageDialog(this, "We decrypt very well <3");
                                    } catch (Exception ex) {
                                        System.err.println(ex.toString());
                                        JOptionPane.showMessageDialog(this, "Something wrong dude </3");
                                    }
                                    break;
                                default:
                                    JOptionPane.showMessageDialog(this, "SELECT A MODE", "OOOOOOOOOOPS", JOptionPane.ERROR_MESSAGE);
                                    break;
                            }break;

                        case 2:
                            switch (jComboBoxMode.getSelectedIndex()) {
                                case 1:
                                    try {
```

```
706.                                        met.decryptDES(rute, "C:\\Users\\josue\\Do
    cuments\\Criptography\\Bloques\\P6NIDIA\\Imagenes\\decrypt_img_ecb_des.bmp", "DES"
    , "DES/ECB/PKCS5Padding ", jTextFieldkey.getText());
707.                                        JOptionPane.showMessageDialog(this, "We de
    crypt very well <3");
708.                                    } catch (Exception ex) {
709.                                        System.err.println(ex.toString());
710.                                        JOptionPane.showMessageDialog(this, "Somet
    hing wrong dude </3");
711.                                    }
712.                                    break;
713.                                case 2:
714.                                    try {
715.                                        met.decryptDES(rute, "C:\\Users\\josue\\Do
    cuments\\Criptography\\Bloques\\P6NIDIA\\Imagenes\\decrypt_img_cbc_des.bmp", "DES"
    , "DES/CBC/PKCS5Padding ", jTextFieldkey.getText(), "01234567");
716.                                        JOptionPane.showMessageDialog(this, "We de
    crypt very well <3");
717.                                    } catch (Exception ex) {
718.                                        System.err.println(ex.toString());
719.                                        JOptionPane.showMessageDialog(this, "Somet
    hing wrong dude </3");
720.                                    }
721.                                    break;
722.                                case 3:
723.                                    try {
724.                                        met.decryptDES(rute, "C:\\Users\\josue\\Do
    cuments\\Criptography\\Bloques\\P6NIDIA\\Imagenes\\decrypt_img_ofb_des.bmp", "DES"
    , "DES/OFB/PKCS5Padding ", jTextFieldkey.getText(), "01234567");
725.                                        JOptionPane.showMessageDialog(this, "We de
    crypt very well <3");
726.                                    } catch (Exception ex) {
727.                                        System.err.println(ex.toString());
728.                                        JOptionPane.showMessageDialog(this, "Somet
    hing wrong dude </3");
729.                                    }
730.                                    break;
731.                                case 4:
732.                                    try {
733.                                        met.decryptDES(rute, "C:\\Users\\josue\\Do
    cuments\\Criptography\\Bloques\\P6NIDIA\\Imagenes\\decrypt_img_cfb_des.bmp", "DES"
    , "DES/CFB/PKCS5Padding ", jTextFieldkey.getText(), "01234567");
734.                                        JOptionPane.showMessageDialog(this, "We de
    crypt very well <3");
735.                                    } catch (Exception ex) {
736.                                        System.err.println(ex.toString());
737.                                        JOptionPane.showMessageDialog(this, "Somet
    hing wrong dude </3");
738.                                    }
739.                                    break;
740.                                default:
741.                                    JOptionPane.showMessageDialog(this, "SELECT MO
    DE", "OOOOOOOOOOOPS", JOptionPane.ERROR_MESSAGE);
742.                                    break;
743.                            }break;
744.                        default:
745.                            JOptionPane.showMessageDialog(this, "SELECT MODE", "OO
    OOOOOOOOPS", JOptionPane.ERROR_MESSAGE);
746.                            break;
747.                    }
748.                }
```

```
749.            }
750.
751.            private void jTextFieldkeyKeyPressed(java.awt.event.KeyEvent evt) {

752.
753.            }
754.
755.            private void jTextFieldkeyKeyTyped(java.awt.event.KeyEvent evt) {

756.
757.            }
758.
759.            private void jTextFieldkeyFocusGained(java.awt.event.FocusEvent evt) {

760.                jTextFieldkey.setText("");
761.            }
762.
763.            private void jComboBoxMethodPropertyChange(java.beans.PropertyChangeEv
    ent evt) {
764.                if(jComboBoxMethod.getSelectedIndex()==1){
765.                    jTextFieldkey.setVisible(true);
766.                    jLabelrestriction.setVisible(true);
767.                    jLabelrestriction.setText("ERROR. KEY LENGHT MUST BE 16 CHARAC
    TERS");
768.                }else if (jComboBoxMethod.getSelectedIndex()==2) {
769.                    jTextFieldkey.setVisible(true);
770.                    jLabelrestriction.setVisible(true);
771.                    jLabelrestriction.setText("ERROR. KEY LENGHT MUST BE 8 CHARACT
    ERS");
772.                }
773.
774.            }
775.
776.            /**
777.             * @param args the command line arguments
778.             */
779.            public static void main(String args[]) {
780.                /* Set the Nimbus look and feel */
781.                //<editor-
    fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">
782.                /* If Nimbus (introduced in Java SE 6) is not available, stay with
     the default look and feel.
783.                 * For details see http://download.oracle.com/javase/tutorial/uisw
    ing/lookandfeel/plaf.html
784.                 */
785.                try {
786.                    for (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.
    UIManager.getInstalledLookAndFeels()) {
787.                        if ("Windows".equals(info.getName())) {
788.                            javax.swing.UIManager.setLookAndFeel(info.getClassName
    ());
789.                            break;
790.                        }
791.                    }
792.                } catch (ClassNotFoundException ex) {
793.                    java.util.logging.Logger.getLogger(Interfaces.class.getName())
    .log(java.util.logging.Level.SEVERE, null, ex);
794.                } catch (InstantiationException ex) {
795.                    java.util.logging.Logger.getLogger(Interfaces.class.getName())
    .log(java.util.logging.Level.SEVERE, null, ex);
796.                } catch (IllegalAccessException ex) {
```

```
797.                    java.util.logging.Logger.getLogger(Interfaces.class.getName())
    .log(java.util.logging.Level.SEVERE, null, ex);
798.                } catch (javax.swing.UnsupportedLookAndFeelException ex) {
799.                    java.util.logging.Logger.getLogger(Interfaces.class.getName())
    .log(java.util.logging.Level.SEVERE, null, ex);
800.                }
801.                //</editor-fold>
802.
803.                /* Create and display the form */
804.                java.awt.EventQueue.invokeLater(new Runnable() {
805.                    public void run() {
806.                        new Interfaces().setVisible(true);
807.                    }
808.                });
809.            }
810.
811.            // Variables declaration - do not modify
812.            private javax.swing.JLabel Fondo;
813.            private javax.swing.ButtonGroup buttonGroup1;
814.            private javax.swing.JButton jButtonED;
815.            private javax.swing.JButton jButtonSelect;
816.            private javax.swing.JComboBox<String> jComboBoxMethod;
817.            private javax.swing.JComboBox<String> jComboBoxMode;
818.            private javax.swing.JLabel jLabel2;
819.            private javax.swing.JLabel jLabel3;
820.            private javax.swing.JLabel jLabelImg;
821.            private javax.swing.JLabel jLabelrestriction;
822.            private javax.swing.JRadioButton jRadioButtonDecrypt;
823.            private javax.swing.JRadioButton jRadioButtonEncrypt;
824.            private javax.swing.JTextField jTextFieldkey;
825.            // End of variables declaration
826.        }
```