|  |  |  |
| --- | --- | --- |
| **ipn** | **INSTITUTO POLITÉCNICO NACIONAL**  **ESCUELA SUPERIOR DE CÓMPUTO** |  |

**Cryptography**

**“Block ciphers”**

Abstact

The following report is an implementation of the block ciphers DES and AES with the four modes they have (ECB,CBC,CFB,OFB) to encrypt and decrypt an image using Java as the programable language.

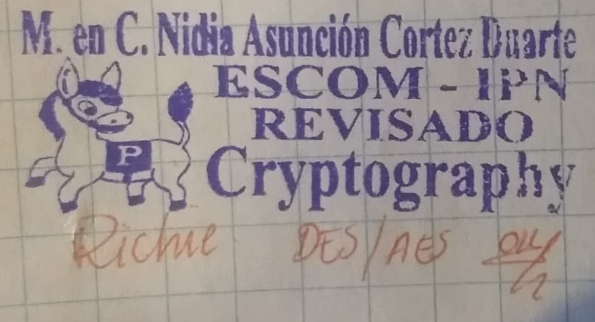
**By:**

**Ricardo Vargas Sagrero**

Professor:

MSc. NIDIA ASUNCIÓN CORTEZ DUARTE

November 2018



To validate this report it is necessary to include the corresponding seal

**Index**

Contenido

[Introducción 1](#_Toc525426585)

[Revisión de literatura 1](#_Toc525426586)

[Software 3](#_Toc525426587)

[Procedimiento 3](#_Toc525426588)

[Resultados 5](#_Toc525426589)

[Discusión 8](#_Toc525426590)

[Conclusión 8](#_Toc525426591)

[Referencias 8](#_Toc525426592)

[Código 9](#_Toc525426593)

# Introduction

Cryptography has its beginnings since the need to hide certain messages from all people, in order that only the sender and receiver could understand its meaning. The origin of the cryptography could be said to have been since writing was created, since with it began to leave written everything that happened in civilizations and so anyone who could read or understand the writing could know what it means. With the time and the generalization of the reading the risks that implied that the information was read by outsiders were seen, for that reason the first methods of protection of the writing were created. Usually the information to be protected is known as a clear message and the hidden information is known as an encrypted message. In this practice we developed a program which is capable of encrypt and decrypt images thanks to the DES algorithm which is implemented in our program with it five modes which are: ECB, CBC, CFB, OFB. The program is designed for computers with Java IDE version 7 or superior installed, we used the following Java Libraries: “java.security.\*” and “java.crypto.\*”, these ones helped us to implement DES algorithm.

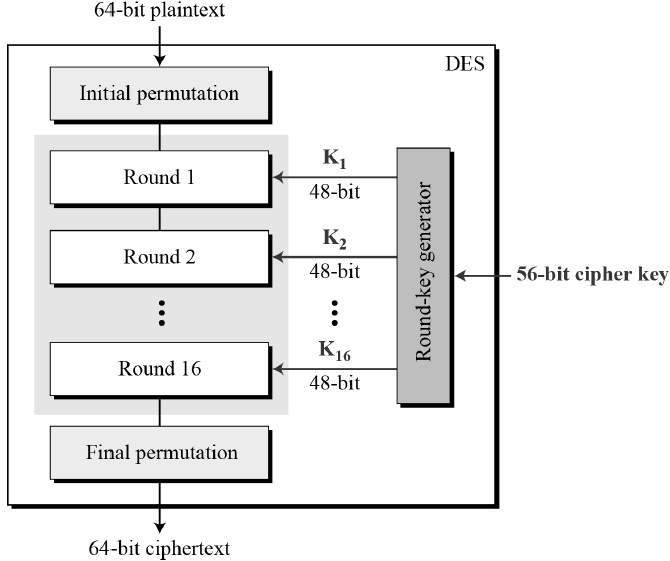
# Literature review

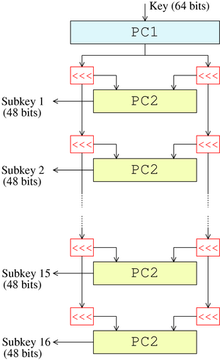
The Data Encryption Standard (DES) is an outdated symmetric-key method of data encryption.

DES works by using the same key to encrypt and decrypt a message, so both the sender and the receiver must know and use the same private key. Once the go-to, symmetric-key algorithm for the encryption of electronic data, DES has been superseded by the more secure Advanced Encryption Standard (AES) algorithm.

The Data Encryption Standard is a block cipher, meaning a cryptographic key and algorithm are applied to a block of data simultaneously rather than one bit at a time. To encrypt a plaintext message, DES groups it into 64-bit blocks. Each block is enciphered using the secret key into a 64-bit ciphertext by means of permutation and substitution. The process involves 16 rounds and can run in four different modes, encrypting blocks individually or making each cipher block dependent on all the previous blocks. Decryption is simply the inverse of encryption, following the same steps but reversing the order in which the keys are applied. For any cipher, the most basic method of attack is brute force, which involves trying each key until you find the right one. The length of the key determines the number of possible keys -- and hence the feasibility -- of this type of attack. DES uses a 64-bit key, but eight of those bits are used for parity checks, effectively limiting the key to 56-bits. Hence, it would take a maximum of 2^56, or 72,057,594,037,927,936, attempts to find the correct key. The Data Encryption Standard (DES) is an outdated symmetric-key method of data encryption.

DES works by using the same key to encrypt and decrypt a message, so both the sender and the receiver must know and use the same private key. Once the go-to, symmetric-key algorithm for the encryption of electronic data, DES has been superseded by the more secure Advanced Encryption Standard (AES) algorithm.

The Data Encryption Standard is a block cipher, meaning a cryptographic key and algorithm are applied to a block of data simultaneously rather than one bit at a time. To encrypt a plaintext message, DES groups it into 64-bit blocks. Each block is enciphered using the secret key into a 64-bit ciphertext by means of permutation and substitution. The process involves 16 rounds and can run in four different modes, encrypting blocks individually or making each cipher block dependent on all the previous blocks. Decryption is simply the inverse of encryption, following the same steps but reversing the order in which the keys are applied. For any cipher, the most basic method of attack is brute force, which involves trying each key until you find the right one. The length of the key determines the number of possible keys -- and hence the feasibility -- of this type of attack. DES uses a 64-bit key, but eight of those bits are used for parity checks, effectively limiting the key to 56-bits. Hence, it would take a maximum of 2^56, or 72,057,594,037,927,936, attempts to find the correct key.

In order to get to understand better how DES algorithm works, and what is the difference between DES and AES, I attached this figure, as we can see, the figure shown in our left, show how DES works, the only differene with AES, is the length of the key, in DES the length is 8 and in AES is 16.

AES is an iterative rather than Feistel cipher. It is based on ‘substitution–permutation network’. It comprises of a series of linked operations, some of which involve replacing inputs by specific outputs (substitutions) and others involve shuffling bits around (permutations).

Interestingly, AES performs all its computations on bytes rather than bits. Hence, AES treats the 128 bits of a plaintext block as 16 bytes. These 16 bytes are arranged in four columns and four rows for processing as a matrix −

Unlike DES, the number of rounds in AES is variable and depends on the length of the key. AES uses 10 rounds for 128-bit keys, 12 rounds for 192-bit keys and 14 rounds for 256-bit keys. Each of these rounds uses a different 128-bit round key, which is calculated from the original AES key.

The schematic of AES structure is given in the following illustration −

Encryption Process

Here, we restrict to description of a typical round of AES encryption. Each round comprise of four sub-processes. The first round process is depicted below −

Byte Substitution (SubBytes)

The 16 input bytes are substituted by looking up a fixed table (S-box) given in design. The result is in a matrix of four rows and four columns.

Shiftrows

Each of the four rows of the matrix is shifted to the left. Any entries that ‘fall off’ are re-inserted on the right side of row. Shift is carried out as follows −

* First row is not shifted.
* Second row is shifted one (byte) position to the left.
* Third row is shifted two positions to the left.
* Fourth row is shifted three positions to the left.
* The result is a new matrix consisting of the same 16 bytes but shifted with respect to each other.

MixColumns

Each column of four bytes is now transformed using a special mathematical function. This function takes as input the four bytes of one column and outputs four completely new bytes, which replace the original column. The result is another new matrix consisting of 16 new bytes. It should be noted that this step is not performed in the last round.

Addroundkey

The 16 bytes of the matrix are now considered as 128 bits and are XORed to the 128 bits of the round key. If this is the last round then the output is the ciphertext. Otherwise, the resulting 128 bits are interpreted as 16 bytes and we begin another similar round.

Decryption Process

The process of decryption of an AES ciphertext is similar to the encryption process in the reverse order. Each round consists of the four processes conducted in the reverse order −

* Add round key
* Mix columns
* Shift rows
* Byte substitution

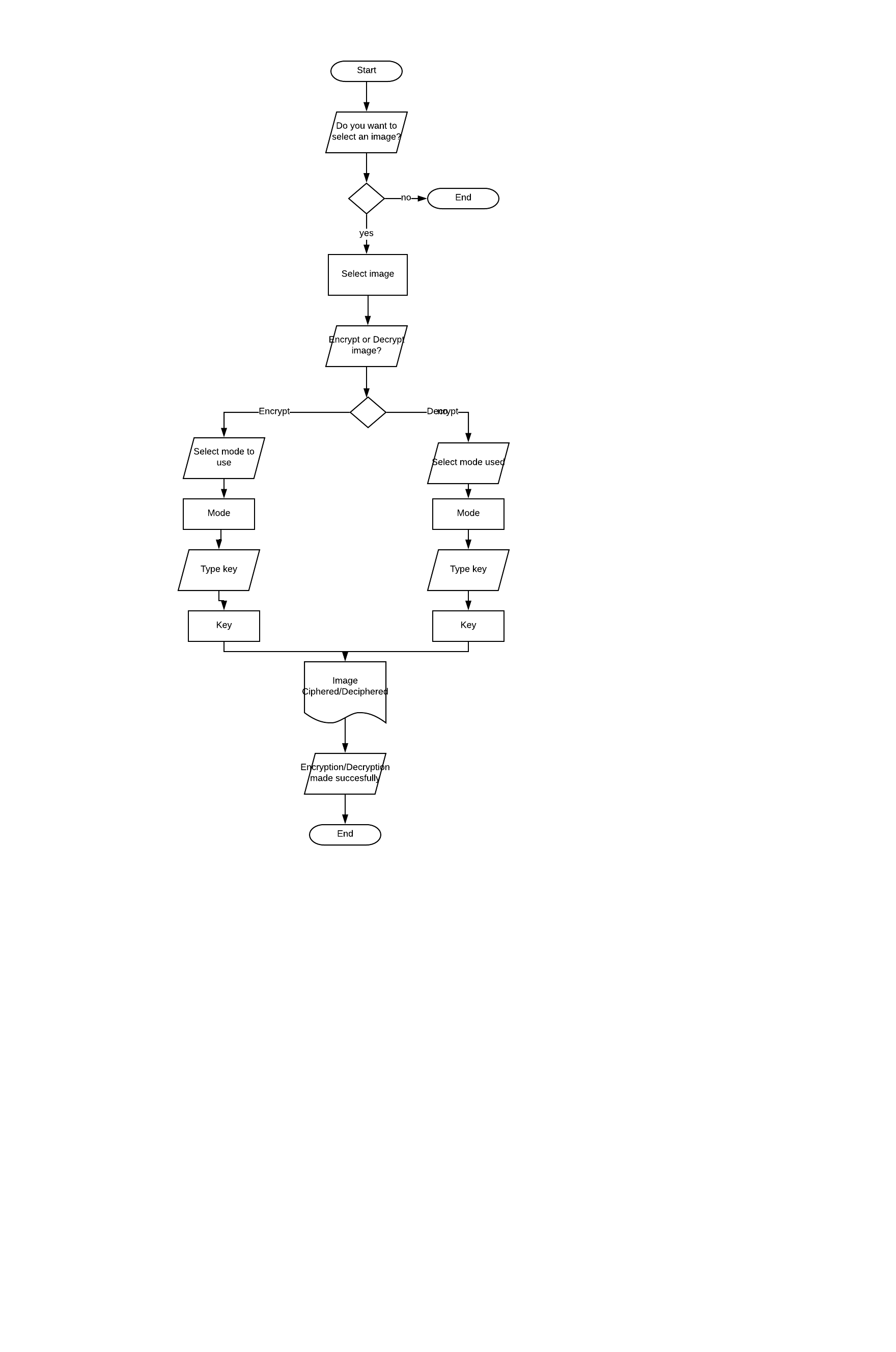
Since sub-processes in each round are in reverse manner, unlike for a Feistel Cipher, the encryption and decryption algorithms needs to be separately implemented, although they are very closely related.

# Software

For this work, the following tools were used

* Integrated development environment: Netbeans
* Programming language: Java
* Operating system: Windows 10
* Standard Java input and output libraries
  + Java AWT
  + Java IO
  + Java Lang
  + Java Security
  + Java Crypto
  + Java Swing

# Procedure

Steps:

1. Open the program.
2. Click “yes” in order to start the encryption/decryption process.
3. Browse and select the image that you want to work with.
4. Select if you want to encrypt or decrypt the image.
   1. For encryption:
      1. Select the mode to use.
      2. Insert the key.
   2. For decryption:
      1. Select the mode used.
      2. Insert the key used.
5. The image is generated (ciphered/deciphered).
6. A message will appear indicating that the process has finished succesfully. Press “Aceptar” in order to finish the process.

# 

# Results

A JFrame was used for this work, resulting in the following:

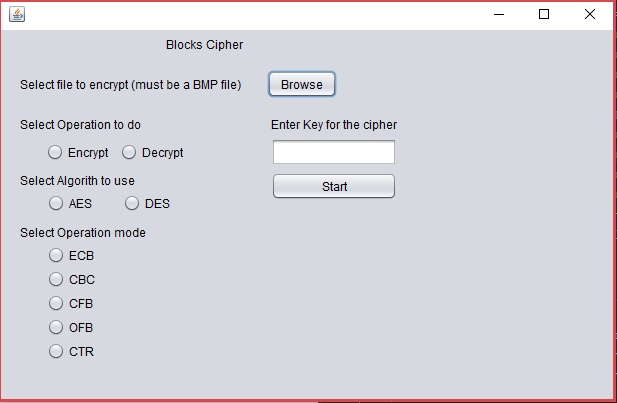


Illustration 1. Initialization of the program

|  |  |  |
| --- | --- | --- |
| Operation Mode | Flag (DES) | Panda (AES) |
| Original Picture |  |  |
| ECB |  |  |
| CBC |  |  |
| OFB |  |  |
| CFB |  |  |

# Discussion

At the beginning I had a couple of questions about the modes of the DES/AES due to one problem I presented was when encrypting two times with the OCB method, the first time encrypting the original image and then encrypting the ciphered image, when doing this, instead of encrypting it, it was deciphered, so I needed to fix the implementation of it, this helped me and make more conscious about testing a lot of times the implementation of algorithms. I had problems with the padding of the image, it had to be 360 times 360 to doesn’t have padding so the decryption process with generate a not corrupt image.

# Conclusion

This practice has been a little tedious cause all the little problems it with the padding, the crypto packages, understanding the operation modes and abstracting all of the BMP structure. This practice helped me a lot to get to understand deeper how cipher method works, how can change a lot from one method from another, and realize how important is to know about cipher methods in order to not only protect other people information but ours, that implementing one good cipher method can make the difference in our program.

# Referencias

1. Theoretically (2015). Affine Cipher - Decryption (Known Plaintext Attack). [video] Available at: https://www.youtube.com/watch?v=ry3g0xN8QKU [Accessed 3 Sep. 2018].
2. Rodriguez, D. (2013). Affine Cipher. [online] Crypto Corner. Available at: http://crypto.interactive-maths.com/affine-cipher.html [Accessed 3 Sep. 2018].
3. Margaret Rouse. (2014). Data Encryption Standard (DES). March 24 2018, de DES Sitio web: <https://searchsecurity.techtarget.com/definition/Data-Encryption-Standard>
4. J. Orlin Grabbe. (2006). The DES Algorithm Illustrated. March 24 2018, de DES Sitio web: <http://page.math.tu-berlin.de/~kant/teaching/hess/krypto-ws2006/des.htm>
5. Tutorials Point . (2004). Advanced Encryption Standard. 25 de abril del 2018, de AES Sitio web: https://www.tutorialspoint.com/cryptography/advanced\_encryption\_standard.htm

# Código

Table . BlockCipherJFrame.java

|  |
| --- |
| 1. /\* 2. \* To change this license header, choose License Headers in Project Properties. 3. \* To change this template file, choose Tools | Templates 4. \* and open the template in the editor. 5. \*/ 6. **package** blockcipher; 8. **import** java.io.File; 9. **import** java.io.FileReader; 10. **import** java.io.IOException; 11. **import** java.io.UnsupportedEncodingException; 12. **import** java.security.InvalidAlgorithmParameterException; 13. **import** java.security.InvalidKeyException; 14. **import** java.security.NoSuchAlgorithmException; 15. **import** java.security.spec.InvalidKeySpecException; 16. **import** java.util.logging.Level; 17. **import** java.util.logging.Logger; 18. **import** javax.crypto.BadPaddingException; 19. **import** javax.crypto.IllegalBlockSizeException; 20. **import** javax.crypto.NoSuchPaddingException; 21. **import** javax.swing.AbstractButton; 22. **import** javax.swing.ButtonGroup; 23. **import** javax.swing.JFileChooser; 24. **import** javax.swing.JOptionPane; 26. /\*\* 27. \* 28. \* @author rsagr 29. \*/ 30. **public** **class** BlockCipherJFrame **extends** javax.swing.JFrame { 32. /\*\* 33. \* Creates new form BlockCipherJFrame 34. \*/ 35. **public** BlockCipherJFrame() { 36. initComponents(); 37. } 39. /\*\* 40. \* This method is called from within the constructor to initialize the form. 41. \* WARNING: Do NOT modify this code. The content of this method is always 42. \* regenerated by the Form Editor. 43. \*/ 44. @SuppressWarnings("unchecked") 45. // <editor-fold defaultstate="collapsed" desc="Generated Code"> 46. **private** **void** initComponents() { 48. buttonGroup1 = **new** javax.swing.ButtonGroup(); 49. buttonGroup2 = **new** javax.swing.ButtonGroup(); 50. buttonGroup3 = **new** javax.swing.ButtonGroup(); 51. jLabel1 = **new** javax.swing.JLabel(); 52. jLabel2 = **new** javax.swing.JLabel(); 53. jButton1 = **new** javax.swing.JButton(); 54. jLabel3 = **new** javax.swing.JLabel(); 55. jRadioButton1 = **new** javax.swing.JRadioButton(); 56. jRadioButton2 = **new** javax.swing.JRadioButton(); 57. jLabel4 = **new** javax.swing.JLabel(); 58. jRadioButton3 = **new** javax.swing.JRadioButton(); 59. jRadioButton4 = **new** javax.swing.JRadioButton(); 60. jLabel5 = **new** javax.swing.JLabel(); 61. jTextField1 = **new** javax.swing.JTextField(); 62. jLabel6 = **new** javax.swing.JLabel(); 63. jRadioButton5 = **new** javax.swing.JRadioButton(); 64. jRadioButton6 = **new** javax.swing.JRadioButton(); 65. jRadioButton7 = **new** javax.swing.JRadioButton(); 66. jRadioButton8 = **new** javax.swing.JRadioButton(); 67. jRadioButton9 = **new** javax.swing.JRadioButton(); 68. jButton2 = **new** javax.swing.JButton(); 70. setDefaultCloseOperation(javax.swing.WindowConstants.EXIT\_ON\_CLOSE); 72. jLabel1.setText("Blocks Cipher "); 74. jLabel2.setText("Select file to encrypt (must be a BMP file)"); 76. jButton1.setText("Browse"); 77. jButton1.addActionListener(**new** java.awt.event.ActionListener() { 78. **public** **void** actionPerformed(java.awt.event.ActionEvent evt) { 79. jButton1ActionPerformed(evt); 80. } 81. }); 83. jLabel3.setText("Select Operation to do"); 85. buttonGroup1.add(jRadioButton1); 86. jRadioButton1.setText("Encrypt"); 87. jRadioButton1.addActionListener(**new** java.awt.event.ActionListener() { 88. **public** **void** actionPerformed(java.awt.event.ActionEvent evt) { 89. jRadioButton1ActionPerformed(evt); 90. } 91. }); 93. buttonGroup1.add(jRadioButton2); 94. jRadioButton2.setText("Decrypt"); 95. jRadioButton2.addActionListener(**new** java.awt.event.ActionListener() { 96. **public** **void** actionPerformed(java.awt.event.ActionEvent evt) { 97. jRadioButton2ActionPerformed(evt); 98. } 99. }); 101. jLabel4.setText("Select Algorith to use"); 103. buttonGroup2.add(jRadioButton3); 104. jRadioButton3.setText("AES"); 105. jRadioButton3.addActionListener(**new** java.awt.event.ActionListener() { 106. **public** **void** actionPerformed(java.awt.event.ActionEvent evt) { 107. jRadioButton3ActionPerformed(evt); 108. } 109. }); 111. buttonGroup2.add(jRadioButton4); 112. jRadioButton4.setText("DES "); 114. jLabel5.setText("Enter Key for the cipher"); 116. jLabel6.setText("Select Operation mode"); 118. buttonGroup3.add(jRadioButton5); 119. jRadioButton5.setText("ECB"); 120. jRadioButton5.addActionListener(**new** java.awt.event.ActionListener() { 121. **public** **void** actionPerformed(java.awt.event.ActionEvent evt) { 122. jRadioButton5ActionPerformed(evt); 123. } 124. }); 126. buttonGroup3.add(jRadioButton6); 127. jRadioButton6.setText("CBC"); 129. buttonGroup3.add(jRadioButton7); 130. jRadioButton7.setText("CFB"); 131. jRadioButton7.addActionListener(**new** java.awt.event.ActionListener() { 132. **public** **void** actionPerformed(java.awt.event.ActionEvent evt) { 133. jRadioButton7ActionPerformed(evt); 134. } 135. }); 137. buttonGroup3.add(jRadioButton8); 138. jRadioButton8.setText("OFB"); 140. buttonGroup3.add(jRadioButton9); 141. jRadioButton9.setText("CTR"); 143. jButton2.setText("Start"); 144. jButton2.addActionListener(**new** java.awt.event.ActionListener() { 145. **public** **void** actionPerformed(java.awt.event.ActionEvent evt) { 146. jButton2ActionPerformed(evt); 147. } 148. }); 150. javax.swing.GroupLayout layout = **new** javax.swing.GroupLayout(getContentPane()); 151. getContentPane().setLayout(layout); 152. layout.setHorizontalGroup( 153. layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING) 154. .addGroup(layout.createSequentialGroup() 155. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING) 156. .addGroup(layout.createSequentialGroup() 157. .addGap(165, 165, 165) 158. .addComponent(jLabel1)) 159. .addGroup(layout.createSequentialGroup() 160. .addGap(19, 19, 19) 161. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING) 162. .addGroup(layout.createSequentialGroup() 163. .addComponent(jLabel2) 164. .addGap(26, 26, 26) 165. .addComponent(jButton1)) 166. .addComponent(jLabel6) 167. .addGroup(layout.createSequentialGroup() 168. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING) 169. .addComponent(jLabel3) 170. .addGroup(layout.createSequentialGroup() 171. .addGap(26, 26, 26) 172. .addComponent(jRadioButton1) 173. .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED) 174. .addComponent(jRadioButton2)) 175. .addComponent(jLabel4) 176. .addGroup(layout.createSequentialGroup() 177. .addGap(27, 27, 27) 178. .addComponent(jRadioButton3) 179. .addGap(31, 31, 31) 180. .addComponent(jRadioButton4))) 181. .addGap(88, 88, 88) 182. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING, **false**) 183. .addComponent(jLabel5, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE) 184. .addComponent(jTextField1) 185. .addComponent(jButton2, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE))))) 186. .addGroup(layout.createSequentialGroup() 187. .addGap(46, 46, 46) 188. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING) 189. .addComponent(jRadioButton6) 190. .addComponent(jRadioButton5) 191. .addComponent(jRadioButton7) 192. .addComponent(jRadioButton8) 193. .addComponent(jRadioButton9)))) 194. .addContainerGap(216, Short.MAX\_VALUE)) 195. ); 196. layout.setVerticalGroup( 197. layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING) 198. .addGroup(layout.createSequentialGroup() 199. .addContainerGap() 200. .addComponent(jLabel1) 201. .addGap(18, 18, 18) 202. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE) 203. .addComponent(jLabel2) 204. .addComponent(jButton1)) 205. .addGap(18, 18, 18) 206. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE) 207. .addComponent(jLabel3) 208. .addComponent(jLabel5)) 209. .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED) 210. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE) 211. .addComponent(jRadioButton1) 212. .addComponent(jRadioButton2) 213. .addComponent(jTextField1, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE)) 214. .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED) 215. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING) 216. .addGroup(layout.createSequentialGroup() 217. .addComponent(jLabel4) 218. .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED) 219. .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE) 220. .addComponent(jRadioButton3) 221. .addComponent(jRadioButton4))) 222. .addComponent(jButton2)) 223. .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED) 224. .addComponent(jLabel6) 225. .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED) 226. .addComponent(jRadioButton5) 227. .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED) 228. .addComponent(jRadioButton6) 229. .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED) 230. .addComponent(jRadioButton7) 231. .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED) 232. .addComponent(jRadioButton8) 233. .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED) 234. .addComponent(jRadioButton9) 235. .addContainerGap(39, Short.MAX\_VALUE)) 236. ); 238. pack(); 239. }// </editor-fold> 241. **private** **void** jButton1ActionPerformed(java.awt.event.ActionEvent evt) { 242. //Browse method we got the file 243. JFileChooser jf = **new** JFileChooser(); 244. **int** a = jf.showOpenDialog(**null**); 245. System.out.println(a); 246. **if**(a == JFileChooser.APPROVE\_OPTION){ 247. **try** { 248. ff = jf.getSelectedFile(); 249. JOptionPane.showMessageDialog(rootPane, "Nombre del archivo \n" + ff.getName()); 251. } **catch** (Exception ex) { 252. Logger.getLogger(BlockCipherJFrame.**class**.getName()).log(Level.SEVERE, **null**, ex); 253. } **finally** { 254. System.out.println(ff.getPath()); 255. } 256. } 257. } 259. **private** **void** jRadioButton1ActionPerformed(java.awt.event.ActionEvent evt) { 260. // TODO add your handling code here: 262. } 264. **private** **void** jRadioButton3ActionPerformed(java.awt.event.ActionEvent evt) { 265. // TODO add your handling code here: 266. } 268. **private** **void** jRadioButton5ActionPerformed(java.awt.event.ActionEvent evt) { 269. // TODO add your handling code here: 270. } 272. **private** **void** jRadioButton7ActionPerformed(java.awt.event.ActionEvent evt) { 273. // TODO add your handling code here: 274. } 276. **private** **void** jRadioButton2ActionPerformed(java.awt.event.ActionEvent evt) { 277. // TODO add your handling code here: 278. } 280. **private** **void** jButton2ActionPerformed(java.awt.event.ActionEvent evt) { 281. //Start method, this will trigger the action for all the variables 282. //Button Group 1 283. jRadioButton1.setActionCommand("-c"); 284. jRadioButton2.setActionCommand("-d"); 285. //Button Group 2 286. jRadioButton3.setActionCommand("2"); 287. jRadioButton4.setActionCommand("1"); 288. //Button Group 3 289. jRadioButton5.setActionCommand("1"); 290. jRadioButton6.setActionCommand("2"); 291. jRadioButton7.setActionCommand("3"); 292. jRadioButton8.setActionCommand("4"); 293. jRadioButton9.setActionCommand("5"); 294. String key = jTextField1.getText(); 295. BlockCipherStruct cipher = **new** BlockCipherStruct(Integer.parseInt(buttonGroup2.getSelection().getActionCommand()),Integer.parseInt(buttonGroup3.getSelection().getActionCommand()),ff,key,buttonGroup1.getSelection().getActionCommand()); 296. **try** { 297. **try** { 298. cipher.cipherAction(); 299. } **catch** (UnsupportedEncodingException ex) { 300. Logger.getLogger(BlockCipherJFrame.**class**.getName()).log(Level.SEVERE, **null**, ex); 301. } **catch** (InvalidKeyException ex) { 302. Logger.getLogger(BlockCipherJFrame.**class**.getName()).log(Level.SEVERE, **null**, ex); 303. } **catch** (IllegalBlockSizeException ex) { 304. Logger.getLogger(BlockCipherJFrame.**class**.getName()).log(Level.SEVERE, **null**, ex); 305. } **catch** (BadPaddingException ex) { 306. Logger.getLogger(BlockCipherJFrame.**class**.getName()).log(Level.SEVERE, **null**, ex); 307. } **catch** (NoSuchAlgorithmException ex) { 308. Logger.getLogger(BlockCipherJFrame.**class**.getName()).log(Level.SEVERE, **null**, ex); 309. } **catch** (InvalidKeySpecException ex) { 310. Logger.getLogger(BlockCipherJFrame.**class**.getName()).log(Level.SEVERE, **null**, ex); 311. } **catch** (NoSuchPaddingException ex) { 312. Logger.getLogger(BlockCipherJFrame.**class**.getName()).log(Level.SEVERE, **null**, ex); 313. } **catch** (InvalidAlgorithmParameterException ex) { 314. Logger.getLogger(BlockCipherJFrame.**class**.getName()).log(Level.SEVERE, **null**, ex); 315. } 316. } **catch** (IOException ex) { 317. Logger.getLogger(BlockCipherJFrame.**class**.getName()).log(Level.SEVERE, **null**, ex); 318. } 319. } 321. /\*\* 322. \* @param args the command line arguments 323. \*/ 324. **public** **static** File ff; 325. **public** **static** **void** main(String args[]) { 326. /\* Set the Nimbus look and feel \*/ 327. //<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) "> 328. /\* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and feel. 329. \* For details see http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html 330. \*/ 331. **try** { 332. **for** (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.getInstalledLookAndFeels()) { 333. **if** ("Nimbus".equals(info.getName())) { 334. javax.swing.UIManager.setLookAndFeel(info.getClassName()); 335. **break**; 336. } 337. } 338. } **catch** (ClassNotFoundException ex) { 339. java.util.logging.Logger.getLogger(BlockCipherJFrame.**class**.getName()).log(java.util.logging.Level.SEVERE, **null**, ex); 340. } **catch** (InstantiationException ex) { 341. java.util.logging.Logger.getLogger(BlockCipherJFrame.**class**.getName()).log(java.util.logging.Level.SEVERE, **null**, ex); 342. } **catch** (IllegalAccessException ex) { 343. java.util.logging.Logger.getLogger(BlockCipherJFrame.**class**.getName()).log(java.util.logging.Level.SEVERE, **null**, ex); 344. } **catch** (javax.swing.UnsupportedLookAndFeelException ex) { 345. java.util.logging.Logger.getLogger(BlockCipherJFrame.**class**.getName()).log(java.util.logging.Level.SEVERE, **null**, ex); 346. } 347. //</editor-fold> 348. /\* Create and display the form \*/ 349. java.awt.EventQueue.invokeLater(**new** Runnable() { 350. **public** **void** run() { 351. **new** BlockCipherJFrame().setVisible(**true**); 352. } 353. }); 354. } 356. // Variables declaration - do not modify 357. **private** javax.swing.ButtonGroup buttonGroup1; 358. **private** javax.swing.ButtonGroup buttonGroup2; 359. **private** javax.swing.ButtonGroup buttonGroup3; 360. **private** javax.swing.JButton jButton1; 361. **private** javax.swing.JButton jButton2; 362. **private** javax.swing.JLabel jLabel1; 363. **private** javax.swing.JLabel jLabel2; 364. **private** javax.swing.JLabel jLabel3; 365. **private** javax.swing.JLabel jLabel4; 366. **private** javax.swing.JLabel jLabel5; 367. **private** javax.swing.JLabel jLabel6; 368. **private** javax.swing.JRadioButton jRadioButton1; 369. **private** javax.swing.JRadioButton jRadioButton2; 370. **private** javax.swing.JRadioButton jRadioButton3; 371. **private** javax.swing.JRadioButton jRadioButton4; 372. **private** javax.swing.JRadioButton jRadioButton5; 373. **private** javax.swing.JRadioButton jRadioButton6; 374. **private** javax.swing.JRadioButton jRadioButton7; 375. **private** javax.swing.JRadioButton jRadioButton8; 376. **private** javax.swing.JRadioButton jRadioButton9; 377. **private** javax.swing.JTextField jTextField1; 378. // End of variables declaration 379. } |

Table . BlockCipherStruct.java

|  |
| --- |
| 1. /\* 2. \* To change this license header, choose License Headers in Project Properties. 3. \* To change this template file, choose Tools | Templates 4. \* and open the template in the editor. 5. \*/ 6. **package** blockcipher; 8. **import** java.awt.image.BufferedImage; 9. **import** java.io.File; 10. **import** java.io.FileOutputStream; 11. **import** java.io.IOException; 12. **import** java.io.UnsupportedEncodingException; 13. **import** **static** java.lang.System.arraycopy; 14. **import** java.nio.file.Files; 15. **import** java.security.InvalidAlgorithmParameterException; 16. **import** java.security.InvalidKeyException; 17. **import** java.security.NoSuchAlgorithmException; 18. **import** java.security.SecureRandom; 19. **import** java.security.spec.InvalidKeySpecException; 20. **import** java.util.Arrays; 21. **import** java.util.Base64; 22. **import** java.util.logging.Level; 23. **import** java.util.logging.Logger; 24. **import** javax.crypto.BadPaddingException; 25. **import** javax.crypto.Cipher; 26. **import** javax.crypto.IllegalBlockSizeException; 27. **import** javax.crypto.NoSuchPaddingException; 28. **import** javax.crypto.SecretKey; 29. **import** javax.crypto.SecretKeyFactory; 30. **import** javax.crypto.spec.DESKeySpec; 31. **import** javax.crypto.spec.IvParameterSpec; 32. **import** javax.crypto.spec.SecretKeySpec; 33. **import** javax.imageio.ImageIO; 34. **import** javax.swing.JOptionPane; 35. **import** org.bouncycastle.jcajce.provider.symmetric.AES; 37. /\*\* 38. \* 39. \* @author rsagr 40. \*/ 41. **public** **class** BlockCipherStruct { 42. **private** BufferedImage gImage; 43. **private** File ff; 44. **private** File newff = **null**; 45. **private** **byte** [] header; 46. **private** **int** index; 47. **private** **int** indexA; 48. **private** String [] modes = {"DES/ECB/PKCS5Padding", 49. "DES/CBC/PKCS5Padding", 50. "DES/CFB/PKCS5Padding", 51. "DES/OFB/PKCS5Padding", //Falta el 8 en OFB8 52. "DES/CTR/PKCS5Padding", 53. "AES/ECB/PKCS5Padding", 54. "AES/CBC/PKCS5Padding", 55. "AES/CFB/PKCS5Padding", 56. "AES/OFB/PKCS5Padding", 57. "AES/CTR/PKCS5Padding"}; 58. **private** String modeSelected = ""; 59. **private** String nameOutFile = ""; 60. **private** String nameAux = ""; 61. **private** String command = ""; 62. **private** String skfS = ""; 63. **private** String key = ""; 64. **private** String finalStringCont; 65. **private** **static** **byte** [] ivDES = {'+','\_','-','b','$','#','=','R'}; 66. **private** **static** **byte** [] ivAES = {'#','F','9',')','(','^',',','/','$','{','5','a','@',':','e','l'}; 67. **public** BlockCipherStruct(**int** indexMode,**int** indexAlgorithm,File f,String keyGiven,String cmd){ 68. index = indexMode; 69. indexA = indexAlgorithm; 70. ff = f; 71. key = keyGiven; 72. command = cmd; 73. } 74. **public** **void** SelectedMode(){ 75. //Select a mode of operation 76. **if**(index == 1){ 77. skfS = "DES"; 78. **switch**(indexA){ 79. **case** 1: 80. modeSelected = modes[0]; 81. nameAux = "DES\_ECB"; 82. **break**; 83. **case** 2: 84. modeSelected = modes[1]; 85. nameAux = "DES\_CBC"; 86. **break**; 87. **case** 3: 88. modeSelected = modes[2]; 89. nameAux = "DES\_CFB"; 90. **break**; 91. **case** 4: 92. modeSelected = modes[3]; 93. nameAux = "DES\_OFB"; 94. **break**; 95. **default**: 96. modeSelected = modes[4]; 97. nameAux = "DES\_CTR"; 98. } 99. } 100. **else**{ 101. skfS = "AES"; 102. **switch**(indexA){ 103. **case** 1: 104. modeSelected = modes[5]; 105. nameAux = "AES\_ECB"; 106. **break**; 107. **case** 2: 108. modeSelected = modes[6]; 109. nameAux = "AES\_CBC"; 110. **break**; 111. **case** 3: 112. modeSelected = modes[7]; 113. nameAux = "AES\_CFB"; 114. **break**; 115. **case** 4: 116. modeSelected = modes[8]; 117. nameAux = "AES\_OFB"; 118. **break**; 119. **default**: 120. modeSelected = modes[9]; 121. nameAux = "AES\_CTR"; 122. **break**; 123. } 124. } 125. System.out.println("Mode Selected: "+modeSelected+"\nnameAux: "+nameAux+"\nKey: "+key); 126. System.out.println("File path"+ff.getPath()); 127. } 128. **private** **void** FileOutName(){ 129. **if**(command.contains("-c")){ 130. nameOutFile = "E"+nameAux+".bmp"; 131. } 132. **else** **if**(command.contains("-d")){ 133. nameOutFile = "D"+nameAux+".bmp"; 134. } 135. System.out.println("Name of the new file: "+nameOutFile); 136. } 137. **private** **byte**[] getHeader(**byte**[] contented){ 138. **byte** [] cab = **new** **byte**[54]; 139. **int** i; 140. **for**(i = 0; i < 54; i++){ 141. cab[i] = contented[i]; 142. } 143. **return** cab; 144. } 145. **private** **byte**[] clrHeader(**byte**[] clr){ 146. **byte** [] aux = **new** **byte**[clr.length-54]; 147. **int** i; 148. **for**(i = 0; i < clr.length-54; i++){ 149. aux[i] = clr[i+54]; 150. } 151. **return** aux; 152. } 153. **private** **void** createImage(**byte**[] c){ 154. **try**{ 155. Files.write(newff.toPath(),c); 156. }**catch**(IOException e){ 157. e.printStackTrace(); 158. } 159. } 160. **private** String strConcat(**byte** [] fS){ 161. //From here we use the cipher to encrypt or decrypt finalString, after doing that we need to contatened the header with the result 162. //using arraycopyFuntion 163. //funtion to print all the bytes from an array 164. //byte[] byteArray = new byte[] { -1, -128, 1, 127 }; 165. //System.out.println(Arrays.toString(byteArray)); 166. **byte**[] contated = **new** **byte**[54+fS.length]; 167. arraycopy(header,0,contated,0,header.length); 168. arraycopy(fS,0,contated,54,fS.length); 169. System.out.println(Arrays.toString(contated)); 170. newff = **new** File(nameOutFile); 171. createImage(contated); 172. **return** finalStringCont; 173. } 174. **private** **byte**[] Action(**byte**[] fString) **throws** UnsupportedEncodingException, NoSuchAlgorithmException, NoSuchPaddingException, InvalidKeyException, InvalidAlgorithmParameterException, IllegalBlockSizeException, BadPaddingException, InvalidKeySpecException{ 175. /\*\*\* 176. \* This is where all the encryption and decryption begins. 177. \* For ECB (Electronic codebook), we don't use a vector so at the 178. \* begging we and a statement to create the vector or not. 179. \* CBC (Cipher-Block Chaining) 180. \* CFB (Cipher FeedBack) 181. \* OFB (Output feedBack) 182. \* CTR (Counter Mode encryption) 183. \*/ 184. **byte** [] encryptedString; 185. System.out.println("Before it goes on the if's: "+nameAux); 186. **if**(nameAux.equals("AES\_ECB") || nameAux.equals("DES\_ECB")){ 187. **byte**[] keyByte = key.getBytes(); 188. SecretKeySpec ss = **new** SecretKeySpec(keyByte,skfS); 189. Cipher ci = Cipher.getInstance(modeSelected); 190. //if index = 1 is encrypt mode 191. **if**(command.contains("-c")){ 192. ci.init(Cipher.ENCRYPT\_MODE, ss); 193. encryptedString = ci.doFinal(fString); 194. **return** encryptedString; 195. } 196. **else**{ 197. ci.init(Cipher.DECRYPT\_MODE,ss); 198. encryptedString = ci.doFinal(fString); 199. **return** encryptedString; 200. } 201. }**else**{ 202. //Statements 203. IvParameterSpec ivspec = **null**; 204. **if**(index == 1){ 205. ivspec = **new** IvParameterSpec(ivDES); 206. } 207. **else**{ 208. System.out.println("Entre a AES para IV  skfS: "+skfS); 209. ivspec = **new** IvParameterSpec(ivAES); 210. } 212. **byte**[] keyByte = key.getBytes(); 213. SecretKeySpec skey = **new** SecretKeySpec(keyByte, skfS); 214. //SecretKeyFactory ss = SecretKeyFactory.getInstance(skfS); 215. //SecretKey securekey = ss.generateSecret(skey); 216. Cipher ci = Cipher.getInstance(modeSelected); 217. **if**(command.contains("-c")){ 218. ci.init(Cipher.ENCRYPT\_MODE,skey,ivspec); 219. encryptedString = ci.doFinal(fString); 220. **return** encryptedString; 221. } 222. **else**{ 223. ci.init(Cipher.DECRYPT\_MODE,skey,ivspec); 224. encryptedString = ci.doFinal(fString); 225. **return** encryptedString; 226. } 227. } 228. } 229. **public** **void** cipherAction() **throws** IOException, UnsupportedEncodingException, InvalidKeyException, IllegalBlockSizeException, BadPaddingException, NoSuchAlgorithmException, InvalidKeySpecException, NoSuchPaddingException, InvalidAlgorithmParameterException{ 230. SelectedMode(); 231. FileOutName(); 232. **byte**[] finalString = Files.readAllBytes(ff.toPath()); 233. header = getHeader(finalString); 234. //System.out.println(Arrays.toString(header)); 235. //System.out.println(Arrays.toString(finalString)); 236. finalString = clrHeader(finalString); 238. /\*Encrypt and decrypt funtion\*/ 239. **byte**[] encryptedString; 240. encryptedString = Action(finalString); 241. System.out.println("final concat"); 242. System.out.println(Arrays.toString(encryptedString)); 243. finalStringCont = strConcat(encryptedString); 245. } 246. **public** BufferedImage getNewImage(File ff){ 247. BufferedImage image = **null**; 248. **try**{ 249. image = ImageIO.read(**new** File(ff.getPath())); 250. }**catch**(IOException e){ 251. JOptionPane.showMessageDialog(**null**,"Error opening the file: "+e); 252. } 253. **return** image; 254. } 256. } |