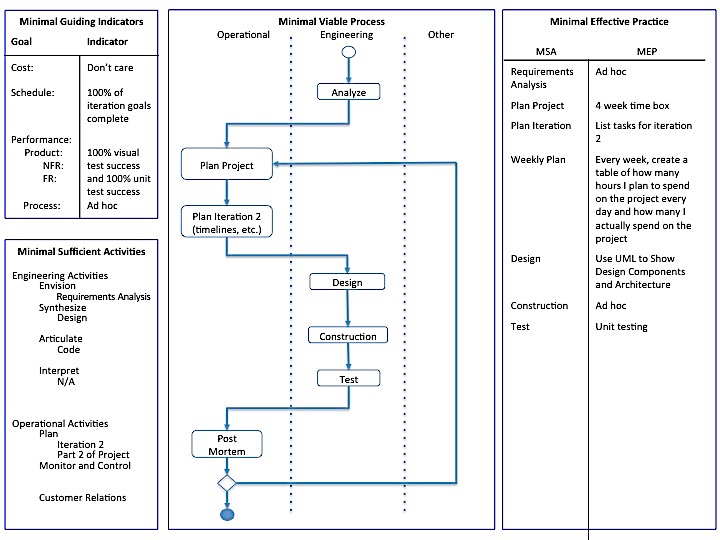
Iteration 6

# Process for Iteration 6



The process has not changed since Iteration 4.

# Requirements Analysis

The requirements have not changed since Iteration 1. They are as follows:

Given a secret image file and two innocent image files, the tool should

* Be able to read in image files and store the pixel information
* Use the extended visual cryptography scheme to encode the secret image pixels into the two innocent images
* Store the encoded images in new image files
  + The filenames and location can be specified by the user. If not, the files are named share1 and share2 and gets stored on the Desktop.

Given two encoded image files, the tool should

* Be able to read in the files and store the pixel information
* Use the extended visual cryptography scheme to decode the secret image from the encoded images (similar to super imposing them)
* The image revealing the secret gets stored in a new image file
  + The filename and location can be specified by the user. If not, the file is named secretMsg and gets stored on the Desktop.

The visual cryptography tool will only work with PNG and JPEG images. The images involved with the encoding process must have the same dimensions. The tool can handle images of any coloring.

# Plans for Project

Iteration 1 (Sept. 11 – Oct. 9):

* Create a graphical user interface
* Get the tool working for strictly black and white images
* Test the tool to check the quality of the encoded shares and the decoded message

Iteration 2 (Oct. 10 – Nov. 6):

* Research visual cryptography schemes with gray scale images

Iteration 3 (Nov. 7 – Dec. 4):

* Implement the visual cryptography scheme with grayscale images
* Begin researching how to modify the current algorithm to handle color images

Iteration 4 (Dec. 5 – Jan. 1):

* Research how to add the ability to encode and decode multicolor images

Iteration 5 (Jan. 2 – Jan. 29):

* Implement the encoding and decoding of color images
* Add unit tests for the components of the visual cryptography tool

Iteration 6 (Jan. 30 – Feb. 26):

* Improve the encryption and decryption process by adding pixel expansion, ie 1 pixel in the cover image becomes 4 pixels in the encoded image
* Analyze the tool and look for ways to improve efficiency (performance and memory storage)
* Add features to project to help boost robustness (i.e. add in checks to keep the user from breaking the tool easily)
* Add unit tests for the components of the visual cryptography tool

# Plans for Iteration 6

* Improve the encryption and decryption process by adding pixel expansion, ie 1 pixel in the cover image becomes 4 pixels in the encoded image
* Analyze the tool and look for ways to improve efficiency (performance and memory storage)
* Add features to project to help boost robustness (i.e. add in checks to keep the user from breaking the tool easily)
* Add unit tests for the components of the visual cryptography tool

# Weekly Plans

Week 1:

|  |  |  |
| --- | --- | --- |
| Day | Expected Hours | Actual Hours |
| Saturday, January 30th | 0 | 0 |
| Sunday, January 31st | 0 | 0 |
| Monday, February 1st | 3 | 1 |
| Tuesday, February 2nd | 3 | 2 |
| Wednesday, February 3rd | 3 | 1 |
| Thursday, February 4th | 0 | 0 |
| Friday, February 5th | 0 | 0 |

Week 2:

|  |  |  |
| --- | --- | --- |
| Day | Expected Hours | Actual Hours |
| Saturday, February 6th | 0 | 0 |
| Sunday, February 7th | 0 | 0 |
| Monday, February 8th | 2 | 1 |
| Tuesday, February 9th | 0 | 2 |
| Wednesday, February 10th | 2 | 1.5 |
| Thursday, February 11th | 0 | 1 |
| Friday, February 12th | 3 | 3 |

Week 3:

|  |  |  |
| --- | --- | --- |
| Day | Expected Hours | Actual Hours |
| Saturday, February 13th | 0 | 4 |
| Sunday, February 14th | 0 | 3 |
| Monday, February 15th | 3 | 1 |
| Tuesday, February 16th | 2 | 3 |
| Wednesday, February 17th | 3 | 2 |
| Thursday, February 18th | 2 | 3.5 |
| Friday, February 19th | 4 | 4 |

Week 4:

|  |  |  |
| --- | --- | --- |
| Day | Expected Hours | Actual Hours |
| Saturday, February 20th | 0 | 0 |
| Sunday, February 21st | 2 | 2 |
| Monday, February 22nd | 3 | 3 |
| Tuesday, February 23rd | 0 | 0 |
| Wednesday, February 24th | 2 | 3 |
| Thursday, February 25th | 2 | 3.5 |
| Friday, February 26th | 0 | 0 |

# Design

Figure 1: UML Diagram from Iteration 1

Note the design has not changed since Iteration 2.

Before changing the Java files to handle the gray scale images, I exported the PlantUML diagram of the visual cryptography tool. Figure 1 shows the class relations.

In Iteration 4, I decided to focus on researching techniques for encrypting color images. The most promising technique I found came from Varalakshmi, R, and Parameswari, and it utilized Visual Information Pixel (VIP) synchronization. VIP synchronization helps hide the secret image pixel information inside the innocent pixels. The process for encrypting a secret image is as follows:

1. Gather and process the two innocent images and one secret image.
2. Half-tone the innocent images using error diffusion.
3. Split the secret image into three images. One image represents only the red concentration of the picture, the second represents the green concentration, and the third represents blue.
4. Perform VIP synchronization on the innocent images and the three secret images.
5. Use error diffusion on the encrypted shares to smooth any pixels that cause the encoded image to look as if they are hiding something.

The testing results from Iteration 5 led me to consider adding pixel expansion to the encryption process. Each pixel in the secret image is represented with four pixels in the encoded images. More details can be found in the construction section.

Decryption for this technique does not require the user to have a computer. The images can be printed on transparencies and stacked to reveal the secret image.

# Construction – NOT DONE YET

The decrypted results at the end of Iteration 5 were not promising. When I had first implemented the VIP synchronization, I tried to avoid the pixel expansion. Since the results were not promising, I decided to put the pixel expansion in the code and perform some visual tests. Now every pixel in the cover and secret images become four pixels in the encoded image.

The first test placed the secret red pixel and secret green pixel with two copies of the first innocent image pixel in encoded image one. The secret blue pixel, along with three copies of the second innocent image pixel, was placed in the second encoded image. With this test case, the secret red, secret green, and secret blue pixels come from the red share, green share, and blue share of the secret image. The tables below give a visual of the pixel expansion.

TABLES

The technique above was improved by performing the VIP synchronization, which splits the red share, green share, and blue share of the secret image into two values that are related to the cover images’ colors. The VIP synchronization technique analyzes the RGB values in their binary format. Below is the pseudo code for the VIP synchronization algorithm. Note, secret[i] refers to a specific digit of the secret color. DOUBLE CHECK THE ALGORITHM

for (every pixel in secret):

if (secret[i] == 1 && coverOne[i] == coverTwo[i]):

Randomly select either coverOne or coverTwo

Make the unselected digit the complement of the selected cover digit

i.e. If coverTwo is selected, then coverOne[i] = !coverTwo[i]

else:

Randomly select either coverOne or coverTwo

Make the unselected digit match the selected over digit

i.e. If coverOne is selected, then coverTwo[i] = coverOne[i]

In my implementation of the code, I broke secret image into three images representing the red, green, and blue concentrations. Therefore, I perform this technique on the colors separately, i.e. the VIP synchronization was performed three separate times.

In the second test, the encoded images had the VIP red, green, and blue pixels along with a cover image pixel. As the tables show below the order of the pixels was not the same for both encoded images.

TABLES

The final test for the VIP synchronization technique during first part of Iteration 6, placed the VIP red, green, and blue pixels with the cover image pixel in the same order for both of the encoded images. Images XX and XX under the Test section were the results from encrypting and decrypting with this last VIP synchronization test.

TABLES

As images XX and XX show, the encoded images are extremely dark. When placing the red, green, or blue VIP results, the other two color concentrations for that pixel were set to 0. This encryption technique utilizes the additive color model, so (0, 0, 0) represents black. Hence, the dark encoded images. Three strategies were tested to fix the darkness: place 255 in the other color concentrations, place 128 in the other color concentrations, and placing the cover image values for the other color concentrations. Based on the test results shown in the next section, I decided to stick with placing 128 in the unspecified concentrations.

Once the darkness problem was fixed, the next hurdle was to add randomness to the ordering of the pixels. When the secret image went through the encryption process, the pixel expansion placed the pixels in the order shown in tables XX and XX. In order to increase the security of the encoded images, I wanted to randomize the order of the pixels in the pixel expansion. Thus, there are twenty-four options for the four pixels to be laid out. This last step finalized the VIP synchronization implementation.

The last modification to be made to the ExtendedVCS class was updating the decryption algorithm. When I was concerned about ensuring the ability to decrypt the secret with transparencies, i.e. physically stacking the images, the pixels of the two encoded images were XOR-ed to get the pixels for the revealed secret. This meant the revealed secret was twice the size of the actual secret message. If the transparency constraint is removed, then I could condense the expanded pixels in the encoded images and make the revealed image the same size as the secret image. My first trial with this scheme involved averaging all the red concentrations, green concentrations, and blue concentrations to find the RGB values to create the color of that pixel. Results are shown in the next section.

Order of tests

X 1/28 – 2/15: Color Image Encryption Testing

X 2/21: Lightening Image Test (for encoded images)

X 2/22: Random Order Testing (randomly order red, green, blue, and cover)

X 2/25 – 3/5: Decryption Testing (Avg Concentration)

# Test – NOT DONE YET

I was testing different images throughout the construction process. The second step of implementation was programming the error diffusion.

|  |
| --- |
|  |
| Figure 2: Original Images (left), Images after Error Diffusion (right) |

Figure 2 shows the results of performing error diffusion on two grayscale images. The puppy pictures appear to be the same, while the rose looks lighter after being processed. The technique worked well enough that, if you only saw the final images, you would not know anything was unusual about the photos.

After being satisfied with the error diffusion, I completed the implementation of the VIP synchronization by adding the code that combined the pixel information of the cover images and the secret image. The first test of the encryption and decryption process was with my grayscale images from Iteration 3. Figures 3 shows the original images. The encoded images are displayed in Figure 4, while Figure 5 shows the secret revealed from stacking the two images in Figure 4.

|  |
| --- |
|  |
| Figure 3: Secret Image to be Encoded (left), Innocent Image 1 (middle), Innocent Image 2 (right) |

|  |
| --- |
|  |
| Figure 4: Encoded Share A (left), Encoded Share B (right) |



Figure 5: Result of decoding shares A and B from Figure 4.

# Post Mortem – NOT DONE YET

At the end of Iteration 6, I began researching other decryption techniques for encrypted images. The

# Source Code

MainFrame.java

1 package Masters\_Proj;  
 2   
 3 /\*  
 4 \* To change this license header, choose License Headers in Project Properties.  
 5 \* To change this template file, choose Tools | Templates  
 6 \* and open the template in the editor.  
 7 \*/  
 8   
 9 /\*\*  
 10 \*  
 11 \* @author allisonholt  
 12 \*/  
 13 public class MainFrame extends javax.swing.JFrame {  
 14   
 15 /\*\*  
 16 \* Creates new form StartFrame  
 17 \*/  
 18 public MainFrame() {  
 19 initComponents();  
 20 this.setLocationRelativeTo(null);  
 21 }  
 22   
 23 /\*\*  
 24 \* This method is called from within the constructor to initialize the form.  
 25 \* WARNING: Do NOT modify this code. The content of this method is always  
 26 \* regenerated by the Form Editor.  
 27 \*/  
 28 @SuppressWarnings("unchecked")  
 29 // <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-BEGIN:initComponents  
 30 private void initComponents() {  
 31   
 32 welcomeBanner = new javax.swing.JLabel();  
 33 jScrollPane1 = new javax.swing.JScrollPane();  
 34 descriptionArea = new javax.swing.JTextArea();  
 35 jScrollPane2 = new javax.swing.JScrollPane();  
 36 directionsArea = new javax.swing.JTextArea();  
 37 encodeButton = new javax.swing.JButton();  
 38 decodeButton = new javax.swing.JButton();  
 39   
 40 setDefaultCloseOperation(javax.swing.WindowConstants.EXIT\_ON\_CLOSE);  
 41 setTitle("Holt Visual Cryptography");  
 42   
 43 welcomeBanner.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);  
 44 welcomeBanner.setText("Welcome to the Holt Visual Cryptography Tool!");  
 45   
 46 descriptionArea.setEditable(false);  
 47 descriptionArea.setColumns(20);  
 48 descriptionArea.setLineWrap(true);  
 49 descriptionArea.setRows(5);  
 50 descriptionArea.setText("The Holt Cryptography Tool allows you to encrypt or decrypt a secret image using extended visual cryptography. The secret image gets embedded into two innocent images that must be superimposed in order to reveal the secret information.");  
 51 descriptionArea.setWrapStyleWord(true);  
 52 jScrollPane1.setViewportView(descriptionArea);  
 53   
 54 directionsArea.setColumns(20);  
 55 directionsArea.setLineWrap(true);  
 56 directionsArea.setRows(5);  
 57 directionsArea.setText("If you wish to encrypt a secret image, then select the encode button. If you wish to decrypt a secret message, then select the decode button.");  
 58 directionsArea.setWrapStyleWord(true);  
 59 jScrollPane2.setViewportView(directionsArea);  
 60   
 61 encodeButton.setText("Encode");  
 62 encodeButton.addActionListener(  
 63 new java.awt.event.ActionListener() {  
 64 public void actionPerformed(java.awt.event.ActionEvent evt) {  
 65 encodePressed(evt);  
 66 }  
 67 });  
 68   
 69 decodeButton.setText("Decode");  
 70 decodeButton.setHorizontalAlignment(javax.swing.SwingConstants.RIGHT);  
 71 decodeButton.addActionListener(  
 72 new java.awt.event.ActionListener() {  
 73 public void actionPerformed(java.awt.event.ActionEvent evt) {  
 74 decodePressed(evt);  
 75 }  
 76 });  
 77   
 78 javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());  
 79 getContentPane().setLayout(layout);  
 80 layout.setHorizontalGroup(  
 81 layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)  
 82 .addComponent(welcomeBanner, javax.swing.GroupLayout.DEFAULT\_SIZE, 600, Short.MAX\_VALUE)  
 83 .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, layout.createSequentialGroup()  
 84 .addContainerGap()  
 85 .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)  
 86 .addComponent(jScrollPane2)  
 87 .addComponent(jScrollPane1))  
 88 .addContainerGap())  
 89 .addGroup(layout.createSequentialGroup()  
 90 .addGap(66, 66, 66)  
 91 .addComponent(encodeButton)  
 92 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)  
 93 .addComponent(decodeButton)  
 94 .addGap(66, 66, 66))  
 95 );  
 96 layout.setVerticalGroup(  
 97 layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)  
 98 .addGroup(layout.createSequentialGroup()  
 99 .addGap(24, 24, 24)  
100 .addComponent(welcomeBanner, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)  
101 .addGap(18, 18, 18)  
102 .addComponent(jScrollPane1, javax.swing.GroupLayout.PREFERRED\_SIZE, 64, javax.swing.GroupLayout.PREFERRED\_SIZE)  
103 .addGap(18, 18, 18)  
104 .addComponent(jScrollPane2, javax.swing.GroupLayout.PREFERRED\_SIZE, 47, javax.swing.GroupLayout.PREFERRED\_SIZE)  
105 .addGap(18, 18, 18)  
106 .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)  
107 .addComponent(encodeButton)  
108 .addComponent(decodeButton))  
109 .addContainerGap(javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE))  
110 );  
111   
112 pack();  
113 }// </editor-fold>//GEN-END:initComponents  
114   
115 private void encodePressed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event\_encodePressed  
116 // TODO add your handling code here:  
117 new EncodeFrame().setVisible(true);  
118 this.setVisible(false);  
119   
120 }//GEN-LAST:event\_encodePressed  
121   
122 private void decodePressed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event\_decodePressed  
123 // TODO add your handling code here:  
124 new DecodeFrame().setVisible(true);  
125 this.setVisible(false);  
126 }//GEN-LAST:event\_decodePressed  
127   
128 /\*\*  
129 \* @param args the command line arguments  
130 \*/  
131 public static void main(String args[]) {  
132 /\* Set the Nimbus look and feel \*/  
133 //<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">  
134 /\* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and feel.  
135 \* For details see http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html   
136 \*/  
137 try {  
138 for (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.getInstalledLookAndFeels()) {  
139 if ("Nimbus".equals(info.getName())) {  
140 javax.swing.UIManager.setLookAndFeel(info.getClassName());  
141 break;  
142 }  
143 }  
144 }   
145 catch (ClassNotFoundException ex) {  
146 java.util.logging.Logger.getLogger(MainFrame.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);  
147 }   
148 catch (InstantiationException ex) {  
149 java.util.logging.Logger.getLogger(MainFrame.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);  
150 }   
151 catch (IllegalAccessException ex) {  
152 java.util.logging.Logger.getLogger(MainFrame.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);  
153 }   
154 catch (javax.swing.UnsupportedLookAndFeelException ex) {  
155 java.util.logging.Logger.getLogger(MainFrame.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);  
156 }  
157 //</editor-fold>  
158 //</editor-fold>  
159   
160 /\* Create and display the form \*/  
161 java.awt.EventQueue.invokeLater(  
162 new Runnable() {  
163 public void run() {  
164 new MainFrame().setVisible(true);  
165 }  
166 });  
167 }  
168   
169 // Variables declaration - do not modify//GEN-BEGIN:variables  
170 private javax.swing.JButton decodeButton;  
171 private javax.swing.JTextArea descriptionArea;  
172 private javax.swing.JTextArea directionsArea;  
173 private javax.swing.JButton encodeButton;  
174 private javax.swing.JScrollPane jScrollPane1;  
175 private javax.swing.JScrollPane jScrollPane2;  
176 private javax.swing.JLabel welcomeBanner;  
177 // End of variables declaration//GEN-END:variables  
178 }  
179

EncodeFrame.java

1 package Masters\_Proj;  
 2   
 3 import java.awt.image.BufferedImage;  
 4 import java.io.File;  
 5 import java.io.IOException;  
 6 import javax.imageio.ImageIO;  
 7 import javax.swing.JFileChooser;  
 8 import javax.swing.JOptionPane;  
 9   
 10 /\*  
 11 \* To change this license header, choose License Headers in Project Properties.  
 12 \* To change this template file, choose Tools | Templates  
 13 \* and open the template in the editor.  
 14 \*/  
 15   
 16 /\*\*  
 17 \*  
 18 \* @author allisonholt  
 19 \*/  
 20 public class EncodeFrame extends javax.swing.JFrame {  
 21   
 22 /\*\*  
 23 \* Creates new form EncodeFrame  
 24 \*/  
 25 public EncodeFrame() {  
 26 initComponents();  
 27 this.setLocationRelativeTo(null);  
 28 }  
 29   
 30 public EncodeFrame(EncodeFrame prevState)  
 31 {  
 32 this.secretTextField.setText(prevState.secretTextField.getText());  
 33 }  
 34   
 35 /\*\*  
 36 \* This method is called from within the constructor to initialize the form.  
 37 \* WARNING: Do NOT modify this code. The content of this method is always  
 38 \* regenerated by the Form Editor.  
 39 \*/  
 40 @SuppressWarnings("unchecked")  
 41 // <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-BEGIN:initComponents  
 42 private void initComponents() {  
 43   
 44 imageChooser = new javax.swing.JFileChooser();  
 45 directoryChooser = new javax.swing.JFileChooser();  
 46 cancelButton = new javax.swing.JButton();  
 47 encodeButton = new javax.swing.JButton();  
 48 panel1 = new javax.swing.JPanel();  
 49 secretTextField = new javax.swing.JTextField();  
 50 jLabel1 = new javax.swing.JLabel();  
 51 browseButton1 = new javax.swing.JButton();  
 52 jPanel1 = new javax.swing.JPanel();  
 53 jLabel2 = new javax.swing.JLabel();  
 54 innocentTextField1 = new javax.swing.JTextField();  
 55 browseButton2 = new javax.swing.JButton();  
 56 innocentTextField2 = new javax.swing.JTextField();  
 57 browseButton3 = new javax.swing.JButton();  
 58 optionalPanel = new javax.swing.JPanel();  
 59 jLabel3 = new javax.swing.JLabel();  
 60 jLabel4 = new javax.swing.JLabel();  
 61 jLabel5 = new javax.swing.JLabel();  
 62 filename1 = new javax.swing.JTextField();  
 63 filename2 = new javax.swing.JTextField();  
 64 jLabel6 = new javax.swing.JLabel();  
 65 storageDirectoryTextField = new javax.swing.JTextField();  
 66 browseButton4 = new javax.swing.JButton();  
 67   
 68 imageChooser.setDialogTitle("Choose an Image");  
 69 imageChooser.setFileFilter(new ImageCustomFilter());  
 70   
 71 directoryChooser.setDialogTitle("Choose a Directory");  
 72 directoryChooser.setFileFilter(new DirectoryCustomFilter());  
 73 directoryChooser.setFileSelectionMode(javax.swing.JFileChooser.DIRECTORIES\_ONLY);  
 74   
 75 setDefaultCloseOperation(javax.swing.WindowConstants.EXIT\_ON\_CLOSE);  
 76 setResizable(false);  
 77   
 78 cancelButton.setText("Cancel");  
 79 cancelButton.addActionListener(  
 80 new java.awt.event.ActionListener() {  
 81 public void actionPerformed(java.awt.event.ActionEvent evt) {  
 82 cancelPressed(evt);  
 83 }  
 84 });  
 85   
 86 encodeButton.setText("Encode");  
 87 encodeButton.addActionListener(  
 88 new java.awt.event.ActionListener() {  
 89 public void actionPerformed(java.awt.event.ActionEvent evt) {  
 90 encodePressed(evt);  
 91 }  
 92 });  
 93   
 94 panel1.setBorder(javax.swing.BorderFactory.createTitledBorder("Secret Image"));  
 95 panel1.setToolTipText("Secret Image");  
 96   
 97 jLabel1.setText("Please select your secret image file:\*");  
 98   
 99 browseButton1.setText("Browse");  
100 browseButton1.addActionListener(  
101 new java.awt.event.ActionListener() {  
102 public void actionPerformed(java.awt.event.ActionEvent evt) {  
103 imageBrowsePressed(evt);  
104 }  
105 });  
106   
107 javax.swing.GroupLayout panel1Layout = new javax.swing.GroupLayout(panel1);  
108 panel1.setLayout(panel1Layout);  
109 panel1Layout.setHorizontalGroup(  
110 panel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)  
111 .addGroup(panel1Layout.createSequentialGroup()  
112 .addComponent(jLabel1)  
113 .addGap(0, 0, Short.MAX\_VALUE))  
114 .addGroup(panel1Layout.createSequentialGroup()  
115 .addComponent(secretTextField)  
116 .addGap(18, 18, 18)  
117 .addComponent(browseButton1))  
118 );  
119 panel1Layout.setVerticalGroup(  
120 panel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)  
121 .addGroup(panel1Layout.createSequentialGroup()  
122 .addContainerGap()  
123 .addComponent(jLabel1)  
124 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)  
125 .addGroup(panel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)  
126 .addComponent(secretTextField, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE)  
127 .addComponent(browseButton1))  
128 .addContainerGap(javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE))  
129 );  
130   
131 jPanel1.setBorder(javax.swing.BorderFactory.createTitledBorder("Innocent Images"));  
132   
133 jLabel2.setText("Please select your two innocent image files:\*");  
134   
135 browseButton2.setText("Browse");  
136 browseButton2.addActionListener(  
137 new java.awt.event.ActionListener() {  
138 public void actionPerformed(java.awt.event.ActionEvent evt) {  
139 imageBrowsePressed(evt);  
140 }  
141 });  
142   
143 browseButton3.setText("Browse");  
144 browseButton3.addActionListener(  
145 new java.awt.event.ActionListener() {  
146 public void actionPerformed(java.awt.event.ActionEvent evt) {  
147 imageBrowsePressed(evt);  
148 }  
149 });  
150   
151 javax.swing.GroupLayout jPanel1Layout = new javax.swing.GroupLayout(jPanel1);  
152 jPanel1.setLayout(jPanel1Layout);  
153 jPanel1Layout.setHorizontalGroup(  
154 jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)  
155 .addGroup(jPanel1Layout.createSequentialGroup()  
156 .addContainerGap()  
157 .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)  
158 .addGroup(jPanel1Layout.createSequentialGroup()  
159 .addComponent(jLabel2)  
160 .addContainerGap(javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE))  
161 .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, jPanel1Layout.createSequentialGroup()  
162 .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)  
163 .addComponent(innocentTextField2, javax.swing.GroupLayout.Alignment.LEADING)  
164 .addComponent(innocentTextField1))  
165 .addGap(18, 18, 18)  
166 .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)  
167 .addComponent(browseButton2)  
168 .addComponent(browseButton3)))))  
169 );  
170 jPanel1Layout.setVerticalGroup(  
171 jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)  
172 .addGroup(jPanel1Layout.createSequentialGroup()  
173 .addContainerGap()  
174 .addComponent(jLabel2)  
175 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)  
176 .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)  
177 .addComponent(innocentTextField1, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE)  
178 .addComponent(browseButton2))  
179 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)  
180 .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)  
181 .addComponent(innocentTextField2, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE)  
182 .addComponent(browseButton3))  
183 .addContainerGap(10, Short.MAX\_VALUE))  
184 );  
185   
186 optionalPanel.setBorder(javax.swing.BorderFactory.createTitledBorder("Optional"));  
187   
188 jLabel3.setText("Names for your encoded shares (without file extension):");  
189   
190 jLabel4.setText("File 1:");  
191   
192 jLabel5.setText("File 2:");  
193   
194 jLabel6.setText("Directory for Image Shares:");  
195   
196 browseButton4.setText("Browse");  
197 browseButton4.addActionListener(  
198 new java.awt.event.ActionListener() {  
199 public void actionPerformed(java.awt.event.ActionEvent evt) {  
200 dirBrowsePressed(evt);  
201 }  
202 });  
203   
204 javax.swing.GroupLayout optionalPanelLayout = new javax.swing.GroupLayout(optionalPanel);  
205 optionalPanel.setLayout(optionalPanelLayout);  
206 optionalPanelLayout.setHorizontalGroup(  
207 optionalPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)  
208 .addGroup(optionalPanelLayout.createSequentialGroup()  
209 .addContainerGap()  
210 .addGroup(optionalPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)  
211 .addGroup(optionalPanelLayout.createSequentialGroup()  
212 .addGroup(optionalPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)  
213 .addComponent(jLabel3)  
214 .addComponent(jLabel6))  
215 .addContainerGap())  
216 .addGroup(optionalPanelLayout.createSequentialGroup()  
217 .addGap(6, 6, 6)  
218 .addGroup(optionalPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)  
219 .addGroup(optionalPanelLayout.createSequentialGroup()  
220 .addComponent(jLabel5)  
221 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)  
222 .addComponent(filename2))  
223 .addGroup(optionalPanelLayout.createSequentialGroup()  
224 .addComponent(jLabel4)  
225 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)  
226 .addComponent(filename1))  
227 .addGroup(optionalPanelLayout.createSequentialGroup()  
228 .addGap(0, 3, Short.MAX\_VALUE)  
229 .addComponent(storageDirectoryTextField, javax.swing.GroupLayout.PREFERRED\_SIZE, 480, javax.swing.GroupLayout.PREFERRED\_SIZE)  
230 .addGap(18, 18, 18)  
231 .addComponent(browseButton4))))))  
232 );  
233 optionalPanelLayout.setVerticalGroup(  
234 optionalPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)  
235 .addGroup(optionalPanelLayout.createSequentialGroup()  
236 .addContainerGap()  
237 .addComponent(jLabel3)  
238 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)  
239 .addGroup(optionalPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)  
240 .addComponent(jLabel4)  
241 .addComponent(filename1, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE))  
242 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)  
243 .addGroup(optionalPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)  
244 .addComponent(jLabel5)  
245 .addComponent(filename2, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE))  
246 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)  
247 .addComponent(jLabel6)  
248 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)  
249 .addGroup(optionalPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)  
250 .addComponent(storageDirectoryTextField, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE)  
251 .addComponent(browseButton4))  
252 .addGap(0, 6, Short.MAX\_VALUE))  
253 );  
254   
255 javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());  
256 getContentPane().setLayout(layout);  
257 layout.setHorizontalGroup(  
258 layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)  
259 .addGroup(layout.createSequentialGroup()  
260 .addContainerGap()  
261 .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)  
262 .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, layout.createSequentialGroup()  
263 .addGap(0, 0, Short.MAX\_VALUE)  
264 .addComponent(encodeButton)  
265 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)  
266 .addComponent(cancelButton))  
267 .addComponent(panel1, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)  
268 .addComponent(jPanel1, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)  
269 .addComponent(optionalPanel, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE))  
270 .addContainerGap())  
271 );  
272 layout.setVerticalGroup(  
273 layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)  
274 .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, layout.createSequentialGroup()  
275 .addContainerGap()  
276 .addComponent(panel1, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE)  
277 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)  
278 .addComponent(jPanel1, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE)  
279 .addGap(12, 12, 12)  
280 .addComponent(optionalPanel, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)  
281 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)  
282 .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)  
283 .addComponent(cancelButton)  
284 .addComponent(encodeButton))  
285 .addContainerGap())  
286 );  
287   
288 pack();  
289 }// </editor-fold>//GEN-END:initComponents  
290   
291 private void cancelPressed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event\_cancelPressed  
292 // TODO add your handling code here:  
293 this.setVisible(false);  
294 new MainFrame().setVisible(true);  
295 }//GEN-LAST:event\_cancelPressed  
296   
297 private void dirBrowsePressed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event\_dirBrowsePressed  
298   
299 int returnVal = directoryChooser.showOpenDialog(this);  
300 if(returnVal == JFileChooser.APPROVE\_OPTION)  
301 {  
302 File dir = directoryChooser.getSelectedFile();  
303 if(evt.getSource() == browseButton4)  
304 {  
305 storageDirectoryTextField.setText(dir.getAbsolutePath());  
306 directoryForStorage = dir.getAbsolutePath();  
307 }  
308 }  
309   
310 }//GEN-LAST:event\_dirBrowsePressed  
311   
312 private void imageBrowsePressed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event\_imageBrowsePressed  
313   
314 int returnVal = imageChooser.showOpenDialog(this);  
315 if(returnVal == JFileChooser.APPROVE\_OPTION)  
316 {  
317 File imageFile = imageChooser.getSelectedFile();  
318 if(evt.getSource() == browseButton1)  
319 {  
320 secretTextField.setText(imageFile.getAbsolutePath());  
321 secretFile = imageFile.getAbsolutePath();  
322 }  
323 else if(evt.getSource() == browseButton2)  
324 {  
325 innocentTextField1.setText(imageFile.getAbsolutePath());  
326 innocentFiles[0] = imageFile.getAbsolutePath();  
327 }  
328 else if(evt.getSource() == browseButton3)  
329 {  
330 innocentTextField2.setText(imageFile.getAbsolutePath());  
331 innocentFiles[1] = imageFile.getAbsolutePath();  
332 }  
333 }  
334 }//GEN-LAST:event\_imageBrowsePressed  
335   
336 private void encodePressed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event\_encodePressed  
337 //Code to encode secret message  
338 BufferedImage secretImage = null;  
339 boolean fileFound;  
340 try  
341 {  
342 secretImage = ImageIO.read(new File(secretFile));  
343 fileFound = true;  
344 }  
345 catch (IOException e)  
346 {  
347 JOptionPane.showMessageDialog(null, "Error reading your secret file",  
348 "ERROR", JOptionPane.ERROR\_MESSAGE);  
349 fileFound = false;  
350 }  
351   
352 BufferedImage[] innocentShares = new BufferedImage[0];  
353 if(fileFound)  
354 {  
355 innocentShares = new BufferedImage[2];  
356 for(int i = 0; i < 2; i++)  
357 {  
358 try  
359 {  
360 innocentShares[i] = ImageIO.read(new File(innocentFiles[i]));  
361 fileFound = true;  
362 }  
363 catch(IOException e)  
364 {  
365 JOptionPane.showMessageDialog(null,   
366 ("Error reading innocent file " + (i + 1)),  
367 "ERROR", JOptionPane.ERROR\_MESSAGE);  
368 fileFound = false;  
369 }  
370 }  
371 }  
372   
373 if(fileFound)  
374 {  
375 ExtendedVCS myEVCS = new ExtendedVCS(secretImage, innocentShares);  
376 myEVCS.encryptImage();  
377   
378 int[][] encodedRGB = myEVCS.getRGBPixelsForShares();  
379   
380 if(storageDirectoryTextField.getText().equals(""))  
381 {  
382 //Get path to users desktop  
383 //BUG!!! Not working.  
384 directoryForStorage = "C:/Users/allisonholt/Desktop";  
385 //makeDir = false;  
386 }  
387   
388 String[] shareFiles = new String[2];  
389   
390 if(filename1.getText().equals(""))  
391 {  
392 shareFiles[0] = directoryForStorage + "/share1.png";  
393 }  
394 else  
395 {  
396 shareFiles[0] = directoryForStorage + "/" + filename1.getText() +".png";  
397 }  
398   
399 if(filename2.getText().equals(""))  
400 {  
401 shareFiles[1] = directoryForStorage + "/share2.png";  
402 }  
403 else  
404 {  
405 shareFiles[1] = directoryForStorage + "/" + filename2.getText() +".png";  
406 }  
407   
408   
409 try  
410 {  
411 BufferedImage tempShare1 = new BufferedImage(myEVCS.getImgWidth(), myEVCS.getImgHeight(), BufferedImage.TYPE\_INT\_RGB);  
412 tempShare1.setRGB(0, 0, myEVCS.getImgWidth(), myEVCS.getImgHeight(), encodedRGB[0], 0, myEVCS.getImgWidth());  
413 File tempOutput1 = new File(shareFiles[0]);  
414 ImageIO.write(tempShare1, "png", tempOutput1);  
415   
416 BufferedImage tempShare2 = new BufferedImage(myEVCS.getImgWidth(), myEVCS.getImgHeight(), BufferedImage.TYPE\_INT\_RGB);  
417 tempShare2.setRGB(0, 0, myEVCS.getImgWidth(), myEVCS.getImgHeight(), encodedRGB[1], 0, myEVCS.getImgWidth());  
418 File tempOutput2 = new File(shareFiles[1]);  
419 ImageIO.write(tempShare2, "png", tempOutput2);  
420   
421 new MainFrame().setVisible(true);  
422 this.setVisible(false);  
423 JOptionPane.showMessageDialog(null, "Your encrypted shares have been created.",  
424 "SUCCESS", JOptionPane.PLAIN\_MESSAGE);  
425 }  
426 catch (IOException e)  
427 {  
428 JOptionPane.showMessageDialog(null, "Error encrypting your secret message",  
429 "ERROR", JOptionPane.ERROR\_MESSAGE);  
430 }  
431   
432 }  
433 }//GEN-LAST:event\_encodePressed  
434   
435 /\*\*  
436 \* @param args the command line arguments  
437 \*/  
438 public static void main(String args[]) {  
439 /\* Set the Nimbus look and feel \*/  
440 //<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">  
441 /\* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and feel.  
442 \* For details see http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html   
443 \*/  
444 try {  
445 for (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.getInstalledLookAndFeels()) {  
446 if ("Nimbus".equals(info.getName())) {  
447 javax.swing.UIManager.setLookAndFeel(info.getClassName());  
448 break;  
449 }  
450 }  
451 }   
452 catch (ClassNotFoundException ex) {  
453 java.util.logging.Logger.getLogger(EncodeFrame.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);  
454 }   
455 catch (InstantiationException ex) {  
456 java.util.logging.Logger.getLogger(EncodeFrame.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);  
457 }   
458 catch (IllegalAccessException ex) {  
459 java.util.logging.Logger.getLogger(EncodeFrame.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);  
460 }   
461 catch (javax.swing.UnsupportedLookAndFeelException ex) {  
462 java.util.logging.Logger.getLogger(EncodeFrame.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);  
463 }  
464 //</editor-fold>  
465   
466 /\* Create and display the form \*/  
467 java.awt.EventQueue.invokeLater(  
468 new Runnable() {  
469 public void run() {  
470 new EncodeFrame().setVisible(true);  
471 }  
472 });  
473 }  
474   
475 //Variables for encoding  
476 private String secretFile = "";  
477 private String[] innocentFiles = new String[2];  
478 private String directoryForStorage = "";  
479   
480 // Variables declaration - do not modify//GEN-BEGIN:variables  
481 private javax.swing.JButton browseButton1;  
482 private javax.swing.JButton browseButton2;  
483 private javax.swing.JButton browseButton3;  
484 private javax.swing.JButton browseButton4;  
485 private javax.swing.JButton cancelButton;  
486 private javax.swing.JFileChooser directoryChooser;  
487 private javax.swing.JButton encodeButton;  
488 private javax.swing.JTextField filename1;  
489 private javax.swing.JTextField filename2;  
490 private javax.swing.JFileChooser imageChooser;  
491 private javax.swing.JTextField innocentTextField1;  
492 private javax.swing.JTextField innocentTextField2;  
493 private javax.swing.JLabel jLabel1;  
494 private javax.swing.JLabel jLabel2;  
495 private javax.swing.JLabel jLabel3;  
496 private javax.swing.JLabel jLabel4;  
497 private javax.swing.JLabel jLabel5;  
498 private javax.swing.JLabel jLabel6;  
499 private javax.swing.JPanel jPanel1;  
500 private javax.swing.JPanel optionalPanel;  
501 private javax.swing.JPanel panel1;  
502 private javax.swing.JTextField secretTextField;  
503 private javax.swing.JTextField storageDirectoryTextField;  
504 // End of variables declaration//GEN-END:variables  
505 }  
506

DecodeFrame.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 Masters\_Proj;  
 7   
 8 import java.awt.image.BufferedImage;  
 9 import java.io.File;  
 10 import java.io.IOException;  
 11 import javax.imageio.ImageIO;  
 12 import javax.swing.JFileChooser;  
 13 import javax.swing.JOptionPane;  
 14   
 15 /\*\*  
 16 \*  
 17 \* @author allisonholt  
 18 \*/  
 19 public class DecodeFrame extends javax.swing.JFrame {  
 20   
 21 /\*\*  
 22 \* Creates new form DecodeFrame  
 23 \*/  
 24 public DecodeFrame() {  
 25 initComponents();  
 26 this.setLocationRelativeTo(null);  
 27 }  
 28   
 29 /\*\*  
 30 \* This method is called from within the constructor to initialize the form.  
 31 \* WARNING: Do NOT modify this code. The content of this method is always  
 32 \* regenerated by the Form Editor.  
 33 \*/  
 34 @SuppressWarnings("unchecked")  
 35 // <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-BEGIN:initComponents  
 36 private void initComponents() {  
 37   
 38 imageChooser = new javax.swing.JFileChooser();  
 39 directoryChooser = new javax.swing.JFileChooser();  
 40 jPanel1 = new javax.swing.JPanel();  
 41 jLabel1 = new javax.swing.JLabel();  
 42 encodedTextField1 = new javax.swing.JTextField();  
 43 browseButton1 = new javax.swing.JButton();  
 44 encodedTextField2 = new javax.swing.JTextField();  
 45 browseButton2 = new javax.swing.JButton();  
 46 jPanel2 = new javax.swing.JPanel();  
 47 jLabel2 = new javax.swing.JLabel();  
 48 jLabel3 = new javax.swing.JLabel();  
 49 stackedTextField = new javax.swing.JTextField();  
 50 jLabel4 = new javax.swing.JLabel();  
 51 storageDirectoryTextField = new javax.swing.JTextField();  
 52 browseButton3 = new javax.swing.JButton();  
 53 jButton2 = new javax.swing.JButton();  
 54 jButton3 = new javax.swing.JButton();  
 55   
 56 imageChooser.setDialogTitle("Choose an Image");  
 57 imageChooser.setFileFilter(new ImageCustomFilter());  
 58   
 59 directoryChooser.setDialogTitle("Choose a Directory");  
 60 directoryChooser.setFileFilter(new DirectoryCustomFilter());  
 61 directoryChooser.setFileSelectionMode(javax.swing.JFileChooser.DIRECTORIES\_ONLY);  
 62   
 63 setDefaultCloseOperation(javax.swing.WindowConstants.EXIT\_ON\_CLOSE);  
 64   
 65 jPanel1.setBorder(javax.swing.BorderFactory.createTitledBorder("Encoded Images"));  
 66   
 67 jLabel1.setText("Please select your two encoded image files:\*");  
 68   
 69 browseButton1.setText("Browse");  
 70 browseButton1.addActionListener(  
 71 new java.awt.event.ActionListener() {  
 72 public void actionPerformed(java.awt.event.ActionEvent evt) {  
 73 imageBrowsePressed(evt);  
 74 }  
 75 });  
 76   
 77 browseButton2.setText("Browse");  
 78 browseButton2.addActionListener(  
 79 new java.awt.event.ActionListener() {  
 80 public void actionPerformed(java.awt.event.ActionEvent evt) {  
 81 imageBrowsePressed(evt);  
 82 }  
 83 });  
 84   
 85 javax.swing.GroupLayout jPanel1Layout = new javax.swing.GroupLayout(jPanel1);  
 86 jPanel1.setLayout(jPanel1Layout);  
 87 jPanel1Layout.setHorizontalGroup(  
 88 jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)  
 89 .addGroup(jPanel1Layout.createSequentialGroup()  
 90 .addContainerGap()  
 91 .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)  
 92 .addGroup(jPanel1Layout.createSequentialGroup()  
 93 .addComponent(jLabel1)  
 94 .addGap(0, 310, Short.MAX\_VALUE))  
 95 .addGroup(jPanel1Layout.createSequentialGroup()  
 96 .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)  
 97 .addComponent(encodedTextField2)  
 98 .addComponent(encodedTextField1))  
 99 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)  
100 .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)  
101 .addComponent(browseButton1)  
102 .addComponent(browseButton2))))  
103 .addContainerGap())  
104 );  
105 jPanel1Layout.setVerticalGroup(  
106 jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)  
107 .addGroup(jPanel1Layout.createSequentialGroup()  
108 .addContainerGap()  
109 .addComponent(jLabel1)  
110 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)  
111 .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)  
112 .addComponent(encodedTextField1, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE)  
113 .addComponent(browseButton1))  
114 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)  
115 .addGroup(jPanel1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)  
116 .addComponent(encodedTextField2, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE)  
117 .addComponent(browseButton2))  
118 .addContainerGap(javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE))  
119 );  
120   
121 jPanel2.setBorder(javax.swing.BorderFactory.createTitledBorder("Optional"));  
122   
123 jLabel2.setText("File Name for Decrypted Secret:");  
124   
125 jLabel3.setText("Name (without extension):");  
126   
127 jLabel4.setText("Directory for Decrypted Image:");  
128   
129 browseButton3.setText("Browse");  
130 browseButton3.addActionListener(  
131 new java.awt.event.ActionListener() {  
132 public void actionPerformed(java.awt.event.ActionEvent evt) {  
133 directoryBrowsePressed(evt);  
134 }  
135 });  
136   
137 javax.swing.GroupLayout jPanel2Layout = new javax.swing.GroupLayout(jPanel2);  
138 jPanel2.setLayout(jPanel2Layout);  
139 jPanel2Layout.setHorizontalGroup(  
140 jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)  
141 .addGroup(jPanel2Layout.createSequentialGroup()  
142 .addContainerGap()  
143 .addGroup(jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)  
144 .addGroup(jPanel2Layout.createSequentialGroup()  
145 .addComponent(jLabel2)  
146 .addGap(0, 0, Short.MAX\_VALUE))  
147 .addGroup(jPanel2Layout.createSequentialGroup()  
148 .addGap(6, 6, 6)  
149 .addGroup(jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)  
150 .addGroup(jPanel2Layout.createSequentialGroup()  
151 .addComponent(jLabel3)  
152 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)  
153 .addComponent(stackedTextField))  
154 .addGroup(jPanel2Layout.createSequentialGroup()  
155 .addComponent(jLabel4)  
156 .addGap(0, 0, Short.MAX\_VALUE))  
157 .addGroup(jPanel2Layout.createSequentialGroup()  
158 .addComponent(storageDirectoryTextField)  
159 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)  
160 .addComponent(browseButton3)))))  
161 .addContainerGap())  
162 );  
163 jPanel2Layout.setVerticalGroup(  
164 jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)  
165 .addGroup(jPanel2Layout.createSequentialGroup()  
166 .addContainerGap()  
167 .addComponent(jLabel2)  
168 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)  
169 .addGroup(jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)  
170 .addComponent(jLabel3)  
171 .addComponent(stackedTextField, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE))  
172 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)  
173 .addGroup(jPanel2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING)  
174 .addGroup(jPanel2Layout.createSequentialGroup()  
175 .addComponent(jLabel4)  
176 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)  
177 .addComponent(storageDirectoryTextField, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE))  
178 .addComponent(browseButton3))  
179 .addContainerGap(19, Short.MAX\_VALUE))  
180 );  
181   
182 jButton2.setText("Cancel");  
183 jButton2.addActionListener(  
184 new java.awt.event.ActionListener() {  
185 public void actionPerformed(java.awt.event.ActionEvent evt) {  
186 cancelPressed(evt);  
187 }  
188 });  
189   
190 jButton3.setText("Decode");  
191 jButton3.addActionListener(  
192 new java.awt.event.ActionListener() {  
193 public void actionPerformed(java.awt.event.ActionEvent evt) {  
194 decodePressed(evt);  
195 }  
196 });  
197   
198 javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());  
199 getContentPane().setLayout(layout);  
200 layout.setHorizontalGroup(  
201 layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)  
202 .addGroup(layout.createSequentialGroup()  
203 .addContainerGap()  
204 .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)  
205 .addComponent(jPanel2, javax.swing.GroupLayout.Alignment.TRAILING, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)  
206 .addComponent(jPanel1, javax.swing.GroupLayout.Alignment.TRAILING, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE)  
207 .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, layout.createSequentialGroup()  
208 .addGap(0, 0, Short.MAX\_VALUE)  
209 .addComponent(jButton3)  
210 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)  
211 .addComponent(jButton2)))  
212 .addContainerGap())  
213 );  
214 layout.setVerticalGroup(  
215 layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)  
216 .addGroup(layout.createSequentialGroup()  
217 .addContainerGap()  
218 .addComponent(jPanel1, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE)  
219 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)  
220 .addComponent(jPanel2, javax.swing.GroupLayout.PREFERRED\_SIZE, javax.swing.GroupLayout.DEFAULT\_SIZE, javax.swing.GroupLayout.PREFERRED\_SIZE)  
221 .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)  
222 .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)  
223 .addComponent(jButton2)  
224 .addComponent(jButton3))  
225 .addContainerGap(javax.swing.GroupLayout.DEFAULT\_SIZE, Short.MAX\_VALUE))  
226 );  
227   
228 pack();  
229 }// </editor-fold>//GEN-END:initComponents  
230   
231 private void cancelPressed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event\_cancelPressed  
232 // TODO add your handling code here:  
233 this.setVisible(false);  
234 new MainFrame().setVisible(true);  
235 }//GEN-LAST:event\_cancelPressed  
236   
237 private void imageBrowsePressed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event\_imageBrowsePressed  
238 // TODO add your handling code here:  
239 int returnVal = imageChooser.showOpenDialog(this);  
240 if(returnVal == JFileChooser.APPROVE\_OPTION)  
241 {  
242 File imageFile = imageChooser.getSelectedFile();  
243 if(evt.getSource() == browseButton1)  
244 {  
245 encodedTextField1.setText(imageFile.getAbsolutePath());  
246 shareFiles[0] = imageFile.getAbsolutePath();  
247 }  
248 else if(evt.getSource() == browseButton2)  
249 {  
250 encodedTextField2.setText(imageFile.getAbsolutePath());  
251 shareFiles[1] = imageFile.getAbsolutePath();  
252 }  
253 }  
254 }//GEN-LAST:event\_imageBrowsePressed  
255   
256 private void directoryBrowsePressed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event\_directoryBrowsePressed  
257 // TODO add your handling code here:  
258 int returnVal = directoryChooser.showOpenDialog(this);  
259 if(returnVal == JFileChooser.APPROVE\_OPTION)  
260 {  
261 File dir = directoryChooser.getSelectedFile();  
262 if(evt.getSource() == browseButton3)  
263 {  
264 storageDirectoryTextField.setText(dir.getAbsolutePath());  
265 directoryForStorage = dir.getAbsolutePath();  
266 }  
267 }  
268 }//GEN-LAST:event\_directoryBrowsePressed  
269   
270 private void decodePressed(java.awt.event.ActionEvent evt) {//GEN-FIRST:event\_decodePressed  
271 // TODO add your handling code here:  
272 BufferedImage[] sharesEVCS = new BufferedImage[2];  
273 boolean fileFound = false;  
274   
275 for(int i = 0; i < 2; i++)  
276 {  
277 try  
278 {  
279 sharesEVCS[i] = ImageIO.read(new File(shareFiles[i]));  
280 fileFound = true;  
281 }  
282 catch(IOException e)  
283 {  
284 JOptionPane.showMessageDialog(null,   
285 ("Error reading file share" + (i + 1)),  
286 "ERROR", JOptionPane.ERROR\_MESSAGE);  
287 fileFound = false;  
288 }  
289 }  
290   
291 if(fileFound)  
292 {  
293 ExtendedVCS myEVCS = new ExtendedVCS(sharesEVCS);  
294 myEVCS.decryptImage();  
295   
296 if(storageDirectoryTextField.getText().equals(""))  
297 {  
298 //Get path to users desktop  
299 //BUG!!! Not working.  
300 directoryForStorage = "C:/Users/allisonholt/Desktop";  
301 //makeDir = false;  
302 }  
303   
304 String decodedFileName;  
305 if(stackedTextField.getText().equals(""))  
306 {  
307 //Get path to users desktop  
308 //BUG!!! Not working.  
309 decodedFileName = directoryForStorage + "/secretMsg.png";  
310 //makeDir = false;  
311 }  
312 else  
313 {  
314 decodedFileName = directoryForStorage + "/" + stackedTextField.getText() + ".png";  
315 }  
316   
317 try  
318 {  
319 BufferedImage decryptImage = new BufferedImage(myEVCS.getImgWidth(), myEVCS.getImgHeight(), BufferedImage.TYPE\_INT\_ARGB);  
320 decryptImage.setRGB(0, 0, myEVCS.getImgWidth(), myEVCS.getImgHeight(), myEVCS.getDecryptImgPixels(), 0, myEVCS.getImgWidth());  
321   
322 File tempOutput = new File(decodedFileName);  
323 ImageIO.write(decryptImage, "png", tempOutput);  
324   
325 new MainFrame().setVisible(true);  
326 this.setVisible(false);  
327 JOptionPane.showMessageDialog(null, "Your decrypted image has been created.",  
328 "SUCCESS", JOptionPane.PLAIN\_MESSAGE);  
329 }  
330 catch(IOException e)  
331 {  
332 JOptionPane.showMessageDialog(null, "Error decrypting your secret message",  
333 "ERROR", JOptionPane.ERROR\_MESSAGE);  
334 }  
335   
336 }  
337 }//GEN-LAST:event\_decodePressed  
338   
339 /\*\*  
340 \* @param args the command line arguments  
341 \*/  
342 public static void main(String args[]) {  
343 /\* Set the Nimbus look and feel \*/  
344 //<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">  
345 /\* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and feel.  
346 \* For details see http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html   
347 \*/  
348 try {  
349 for (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.getInstalledLookAndFeels()) {  
350 if ("Nimbus".equals(info.getName())) {  
351 javax.swing.UIManager.setLookAndFeel(info.getClassName());  
352 break;  
353 }  
354 }  
355 }   
356 catch (ClassNotFoundException ex) {  
357 java.util.logging.Logger.getLogger(DecodeFrame.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);  
358 }   
359 catch (InstantiationException ex) {  
360 java.util.logging.Logger.getLogger(DecodeFrame.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);  
361 }   
362 catch (IllegalAccessException ex) {  
363 java.util.logging.Logger.getLogger(DecodeFrame.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);  
364 }   
365 catch (javax.swing.UnsupportedLookAndFeelException ex) {  
366 java.util.logging.Logger.getLogger(DecodeFrame.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);  
367 }  
368 //</editor-fold>  
369   
370 /\* Create and display the form \*/  
371 java.awt.EventQueue.invokeLater(  
372 new Runnable() {  
373 public void run() {  
374 new DecodeFrame().setVisible(true);  
375 }  
376 });  
377 }  
378 //Variables for decoding  
379 private String[] shareFiles = new String[2];  
380 private String directoryForStorage = "";  
381   
382 // Variables declaration - do not modify//GEN-BEGIN:variables  
383 private javax.swing.JButton browseButton1;  
384 private javax.swing.JButton browseButton2;  
385 private javax.swing.JButton browseButton3;  
386 private javax.swing.JFileChooser directoryChooser;  
387 private javax.swing.JTextField encodedTextField1;  
388 private javax.swing.JTextField encodedTextField2;  
389 private javax.swing.JFileChooser imageChooser;  
390 private javax.swing.JButton jButton2;  
391 private javax.swing.JButton jButton3;  
392 private javax.swing.JLabel jLabel1;  
393 private javax.swing.JLabel jLabel2;  
394 private javax.swing.JLabel jLabel3;  
395 private javax.swing.JLabel jLabel4;  
396 private javax.swing.JPanel jPanel1;  
397 private javax.swing.JPanel jPanel2;  
398 private javax.swing.JTextField stackedTextField;  
399 private javax.swing.JTextField storageDirectoryTextField;  
400 // End of variables declaration//GEN-END:variables  
401 }  
402

ImageCustomFilter.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 Masters\_Proj;  
 7   
 8 import java.io.File;  
 9   
10 /\*\*  
11 \*  
12 \* @author allisonholt  
13 \*/  
14 public class ImageCustomFilter extends javax.swing.filechooser.FileFilter {  
15   
16 @Override  
17 public boolean accept(File file)  
18 {  
19 //allow only image file  
20 return file.isDirectory() || file.getAbsolutePath().endsWith(".png")  
21 || file.getAbsolutePath().endsWith(".jpeg")  
22 || file.getAbsolutePath().endsWith(".jpg");  
23 }  
24   
25 @Override  
26 public String getDescription()  
27 {  
28 return "Image files (\*.png, \*.jpeg, \*.jpg)";  
29 }  
30   
31 }  
32

DirectoryCustomFilter.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 Masters\_Proj;  
 7   
 8 import java.io.File;  
 9   
10 /\*\*  
11 \*  
12 \* @author allisonholt  
13 \*/  
14 public class DirectoryCustomFilter extends javax.swing.filechooser.FileFilter{  
15   
16 @Override  
17 public boolean accept(File file)  
18 {  
19 //allow only image file  
20 return file.isDirectory();  
21 }  
22   
23 @Override  
24 public String getDescription()  
25 {  
26 return "File Directory";  
27 }  
28 }  
29

ExtendedVCS.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 Masters\_Proj;  
 7 import java.awt.Color;  
 8 import java.awt.image.BufferedImage;  
 9 import java.util.Arrays;  
 10 import java.util.ArrayList;  
 11 import java.util.Collections;  
 12 import java.util.Random;  
 13   
 14 /\*\*  
 15 \* This class contains all the methods needed to  
 16 \* perform the necessary operations for visual  
 17 \* cryptography.  
 18 \*   
 19 \* @author Allison Holt  
 20 \* @version 02-21-2016  
 21 \*/  
 22 public class ExtendedVCS   
 23 {  
 24 private int k;  
 25 private int n;  
 26 private int imgWidth;  
 27 private int imgHeight;  
 28 private int numColumns;  
 29 private BufferedImage secretMsg;  
 30 private BufferedImage[] innocentShares;  
 31 //private int[2][] shareOrigRGBPixels;  
 32 private int[][] encryptedShareRGB;  
 33 private int[][] secretSharesRGB;  
 34   
 35 private int numSharesToDecrypt;  
 36 private BufferedImage[] sharesToDecrypt;  
 37 private int[] secretMsgPixels;  
 38   
 39   
 40 /\*\*  
 41 \* This version of the constructor is meant to perform the encryption process.  
 42 \*   
 43 \* @param secretMsgIn The image to be encoded into the cover images.  
 44 \* @param innocentSharesIn The array contains the two cover images for the encryption process.  
 45 \*/  
 46 public ExtendedVCS(BufferedImage secretMsgIn, BufferedImage[] innocentSharesIn)  
 47 {  
 48 k = 2;  
 49 n = 2;  
 50 secretMsg = secretMsgIn;  
 51 imgWidth = secretMsg.getWidth();  
 52 imgHeight = secretMsg.getHeight();  
 53 innocentShares = innocentSharesIn;  
 54 encryptedShareRGB = new int[2][imgWidth \* imgHeight];  
 55 }  
 56   
 57 //For decryption purposes  
 58 /\*\*  
 59 \* This version of the constructor is meant to perform the decryption process.  
 60 \*   
 61 \* @param shareImgs The array contains two encoded images to be stacked and the secret decrypted.  
 62 \*/  
 63 public ExtendedVCS(BufferedImage[] shareImgs)  
 64 {  
 65 numSharesToDecrypt = 2;  
 66 sharesToDecrypt = shareImgs;  
 67 imgWidth = shareImgs[0].getWidth();  
 68 imgHeight = shareImgs[0].getHeight();  
 69 }  
 70   
 71 /\*\*  
 72 \* Method returns the width of the images.  
 73 \*  
 74 \* @return The width of the images used in the encryption/decryption.  
 75 \*/  
 76 public int getImgWidth()  
 77 {  
 78 return imgWidth;  
 79 }  
 80   
 81 /\*\*  
 82 \* Method returns the height of the images.  
 83 \*  
 84 \* @return The height of the images used in the encryption/decryption.  
 85 \*/  
 86 public int getImgHeight()  
 87 {  
 88 return imgHeight;  
 89 }  
 90   
 91 /\*\*  
 92 \* Method returns the pixels for the encoded images.  
 93 \*  
 94 \* @return The 2D array containing the pixel information for both encoded images.  
 95 \*/  
 96 public int[][] getRGBPixelsForShares()  
 97 {  
 98 return encryptedShareRGB;  
 99 }  
100   
101 /\*\*  
102 \* Method returns the pixels of the decrypted secret.  
103 \*  
104 \* @return The array containing the pixel information for the decrypted secret image.  
105 \*/  
106 public int[] getDecryptImgPixels()  
107 {  
108 return secretMsgPixels;  
109 }  
110   
111 /\*\*  
112 \* Method that orchestrates the encryption process and calls the helper  
113 \* methods necessary.  
114 \*/  
115 public void encryptImage()  
116 {  
117 int[] secretRGB = new int[imgWidth \* imgHeight];  
118 //A cover image is the same as an innocent image  
119 int[][] coverRGB = new int[2][imgWidth \* imgHeight];  
120   
121 //Process the gathered innocent images and the secret image  
122 secretMsg.getRGB(0, 0, imgWidth, imgHeight, secretRGB, 0, imgWidth);  
123 innocentShares[0].getRGB(0, 0, imgWidth, imgHeight, coverRGB[0], 0, imgWidth);  
124 innocentShares[1].getRGB(0, 0, imgWidth, imgHeight, coverRGB[1], 0, imgWidth);  
125   
126 //Half-tone Innocent Images  
127 errorDiffusion(coverRGB[0]);  
128 errorDiffusion(coverRGB[1]);  
129   
130 //Split secret image into three images  
131 int[] secretRed = new int[secretRGB.length];  
132 int[] secretGreen = new int[secretRGB.length];  
133 int[] secretBlue = new int[secretRGB.length];  
134 splitSecretRGB(secretRGB, secretRed, secretGreen, secretBlue);  
135   
136 //VIP synchronization  
137 vipSynchronization(secretRed, secretGreen, secretBlue, coverRGB);  
138   
139 //Perform error diffusion on cover images with secret encoded  
140 errorDiffusion(encryptedShareRGB[0]);  
141 errorDiffusion(encryptedShareRGB[1]);  
142   
143 }  
144   
145 /\*\*  
146 \* Method utilizes the Floyd-Steinberg dithering technique for blending  
147 \* the pixels together for a more continuous look.  
148 \*  
149 \* @param secret The array containing the pixels of an image.  
150 \*/  
151 private void errorDiffusion(int[] image)  
152 {  
153 int x[][] = new int[imgHeight][imgWidth];  
154 int u[][] = new int [imgHeight][imgWidth];  
155   
156 int i = 0;  
157 for(int n = 0; n < imgHeight; n++)  
158 {  
159 for(int m = 0; m < imgWidth; m++)  
160 {  
161 x[n][m] = image[i];  
162 i += 1;  
163 }  
164 }  
165   
166 for(int n = 0; n < imgHeight; n++)  
167 {  
168 for(int m = 0; m < imgWidth; m++)  
169 {  
170 u[n][m] += x[n][m];  
171   
172 int xRed = (x[n][m] & 0x00ff0000) >> 16;  
173 int xGreen = (x[n][m] & 0x0000ff00) >> 8;  
174 int xBlue = (x[n][m] & 0x000000ff);  
175   
176 int uRed = (u[n][m] & 0x00ff0000) >> 16;  
177 int uGreen = (u[n][m] & 0x0000ff00) >> 8;  
178 int uBlue = (u[n][m] & 0x000000ff);  
179   
180 int quantErrorRed = uRed - xRed;  
181 int quantErrorGreen = uGreen - xGreen;  
182 int quantErrorBlue = uBlue - xBlue;  
183   
184 if(xRed > 127)  
185 {  
186 if((m + 1) < imgWidth)  
187 {  
188 int temp = quantErrorRed \* 7 / 16;  
189 temp = temp << 16;  
190 u[n][m + 1] += temp;  
191 }  
192 if((m - 1) >= 0 && (n + 1) < imgHeight)  
193 {  
194 int temp = quantErrorRed \* 3 / 16;  
195 temp = temp << 16;  
196 u[n + 1][m - 1] += temp;  
197 }  
198 if((n + 1) < imgHeight)  
199 {  
200 int temp = quantErrorRed \* 5 / 16;  
201 temp = temp << 16;  
202 u[n + 1][m] += temp;  
203 }  
204 if((m + 1) < imgWidth && (n + 1) < imgHeight)  
205 {  
206 int temp = quantErrorRed \* 1 / 16;  
207 temp = temp << 16;  
208 u[n + 1][m + 1] += temp;  
209 }  
210 }  
211 if(xGreen > 127)  
212 {  
213 if((m + 1) < imgWidth)  
214 {  
215 int temp = quantErrorGreen \* 7 / 16;  
216 temp = temp << 8;  
217 u[n][m + 1] += temp;  
218 }  
219 if((m - 1) >= 0 && (n + 1) < imgHeight)  
220 {  
221 int temp = quantErrorGreen \* 3 / 16;  
222 temp = temp << 8;  
223 u[n + 1][m - 1] += temp;  
224 }  
225 if((n + 1) < imgHeight)  
226 {  
227 int temp = quantErrorGreen \* 5 / 16;  
228 temp = temp << 8;  
229 u[n + 1][m] += temp;  
230 }  
231 if((m + 1) < imgWidth && (n + 1) < imgHeight)  
232 {  
233 int temp = quantErrorGreen \* 1 / 16;  
234 temp = temp << 8;  
235 u[n + 1][m + 1] += temp;  
236 }  
237 }  
238 if(xBlue > 127)  
239 {  
240 if((m + 1) < imgWidth)  
241 {  
242 int temp = quantErrorBlue \* 7 / 16;  
243 u[n][m + 1] += temp;  
244 }  
245 if((m - 1) >= 0 && (n + 1) < imgHeight)  
246 {  
247 int temp = quantErrorBlue \* 3 / 16;  
248 u[n + 1][m - 1] += temp;  
249 }  
250 if((n + 1) < imgHeight)  
251 {  
252 int temp = quantErrorBlue \* 5 / 16;  
253 u[n + 1][m] += temp;  
254 }  
255 if((m + 1) < imgWidth && (n + 1) < imgHeight)  
256 {  
257 int temp = quantErrorBlue \* 1 / 16;  
258 u[n + 1][m + 1] += temp;  
259 }  
260 }  
261 }  
262 }  
263   
264 int j = 0;  
265 for(int n = 0; n < imgHeight; n++)  
266 {  
267 for(int m = 0; m < imgWidth; m++)  
268 {  
269 image[j] = u[n][m];  
270 j += 1;  
271 }  
272 }  
273 }  
274   
275 /\*\*  
276 \* Method breaks the secret image is broken up into three separate images  
277 \* based on the red, green, and blue concentrations.  
278 \*  
279 \* @param secret The 2D array containing the pixels of the secret images.  
280 \* @param red The red concentration of each pixel in the secret message.  
281 \* @param green The green concentration of each pixel in the secret message.  
282 \* @param blue The blue concentration of each pixel in the secret message.  
283 \*/  
284 private void splitSecretRGB(int[] secret, int[] red, int[] green, int[] blue)  
285 {  
286 for(int i = 0; i < secret.length; i++)  
287 {  
288 int redVal = (secret[i] & 0x00ff0000) >> 16;  
289 int greenVal = (secret[i] & 0x0000ff00) >> 8;  
290 int blueVal = (secret[i] & 0x000000ff);  
291   
292 Pixel redPix = new Pixel(redVal, 0, 0);  
293 Pixel greenPix = new Pixel(0, greenVal, 0);  
294 Pixel bluePix = new Pixel (0, 0, blueVal);  
295   
296 int redCon = redPix.getConcentration('r');  
297 int greenCon = greenPix.getConcentration('g');  
298 int blueCon = bluePix.getConcentration('b');  
299   
300 Color redColor = new Color(redCon, 0, 0);  
301 Color greenColor = new Color(0, greenCon, 0);  
302 Color blueColor = new Color(0, 0, blueCon);  
303   
304 red[i] = redColor.getRGB();  
305 green[i] = greenColor.getRGB();  
306 blue[i] = blueColor.getRGB();  
307 }  
308 }  
309   
310 /\*\*  
311 \* Method takes the color shares of the secret message and the pixels of the  
312 \* cover images and combines them so the cover images don't lose their meaning  
313 \* while encoding the secret message. The process does perform pixel expansion,  
314 \* i.e. a single pixel gets represented by four in the encoded image. The order  
315 \* of the pixels (red, green, blue, or cover) get shuffled with every pixel.  
316 \*  
317 \* @param red The red concentration of each pixel in the secret message.  
318 \* @param green The green concentration of each pixel in the secret message.  
319 \* @param blue The blue concentration of each pixel in the secret message.  
320 \* @param cover The 2D array containing the pixels of the cover images.  
321 \*/  
322 private void vipSynchronization(int[] red, int[] green, int[] blue, int[][] cover)  
323 {  
324 int[][] cover1 = new int[imgHeight][imgWidth];  
325 int[][] cover2 = new int[imgHeight][imgWidth];  
326   
327 int[][] encoded1 = new int[imgHeight \* 2][imgWidth \* 2];  
328 int[][] encoded2 = new int[imgHeight \* 2][imgWidth \* 2];  
329   
330 ArrayList<String> colorOrder = new ArrayList<String>();  
331 Collections.addAll(colorOrder, "red", "green", "blue", "cover");  
332   
333 int n = 0;  
334 for(int i = 0; i < imgHeight; i++)  
335 {  
336 for(int j = 0; j < imgWidth; j++)  
337 {  
338 cover1[i][j] = cover[0][n];  
339 cover2[i][j] = cover[1][n];  
340 n++;  
341 }  
342 }  
343   
344 for(int i = 0; i < cover[0].length; i++)  
345 {  
346 int c1Red = (cover[0][i] & 0x00ff0000) >> 16;  
347 int c2Red = (cover[1][i] & 0x00ff0000) >> 16;  
348 int secretRed = (red[i] & 0x00ff0000) >> 16;  
349 String c1RedBinary = String.format("%8s", Integer.toBinaryString(c1Red)).replace(" ", "0");  
350 String c2RedBinary = String.format("%8s", Integer.toBinaryString(c2Red)).replace(" ", "0");  
351 String secretRedBinary = String.format("%8s",Integer.toBinaryString(secretRed)).replace(" ", "0");  
352   
353 for(int j = 0; j < secretRedBinary.length(); j++)  
354 {  
355 if(secretRedBinary.charAt(j) == '1'  
356 && c1RedBinary.charAt(j) == c2RedBinary.charAt(j))  
357 {  
358 Random rand = new Random();  
359 int temp = rand.nextInt(20) % 2;  
360 /\*  
361 if temp == 0 then c1 stays the same and c2 is flipped  
362 if temp == 1 then c2 stays the same and c1 is flipped  
363 \*/  
364 if(temp == 0 && c2RedBinary.charAt(j) == '1')  
365 {  
366 char[] c2Array = c2RedBinary.toCharArray();  
367 c2Array[j] = '0';  
368 c2RedBinary = new String(c2Array);  
369 }  
370 else if(temp == 0 && c2RedBinary.charAt(j) == '0')  
371 {  
372 char[] c2Array = c2RedBinary.toCharArray();  
373 c2Array[j] = '1';  
374 c2RedBinary = new String(c2Array);  
375 }  
376 else if(temp == 1 && c1RedBinary.charAt(j) == '1')  
377 {  
378 char[] c1Array = c1RedBinary.toCharArray();  
379 c1Array[j] = '0';  
380 c1RedBinary = new String(c1Array);  
381 }  
382 else  
383 {  
384 char[] c1Array = c1RedBinary.toCharArray();  
385 c1Array[j] = '1';  
386 c1RedBinary = new String(c1Array);  
387 }  
388 }  
389 else  
390 {  
391 Random rand = new Random();  
392 int temp = rand.nextInt(20) % 2;  
393 /\*  
394 if temp == 0 then c2 bit is set to c1 bit  
395 if temp == 1 then c1 bit is set to c2 bit  
396 \*/  
397 if(temp == 0)  
398 {  
399 char[] c2Array = c2RedBinary.toCharArray();  
400 c2Array[j] = c1RedBinary.charAt(j);  
401 c2RedBinary = new String(c2Array);  
402 }  
403 else  
404 {  
405 char[] c1Array = c1RedBinary.toCharArray();  
406 c1Array[j] = c2RedBinary.charAt(j);  
407 c1RedBinary = new String(c1Array);  
408 }  
409 }  
410 }  
411   
412 int c1Green = (cover[0][i] & 0x0000ff00) >> 8;  
413 int c2Green = (cover[1][i] & 0x0000ff00) >> 8;  
414 int secretGreen = (green[i] & 0x0000ff00) >> 8;  
415 String c1GreenBinary = String.format("%8s", Integer.toBinaryString(c1Green)).replace(" ", "0");  
416 String c2GreenBinary = String.format("%8s", Integer.toBinaryString(c2Green)).replace(" ", "0");  
417 String secretGreenBinary = String.format("%8s", Integer.toBinaryString(secretGreen)).replace(" ", "0");  
418   
419 for(int j = 0; j < secretGreenBinary.length(); j++)  
420 {  
421 if(secretGreenBinary.charAt(j) == '1'  
422 && c1GreenBinary.charAt(j) == c2GreenBinary.charAt(j))  
423 {  
424 Random rand = new Random();  
425 int temp = rand.nextInt(20) % 2;  
426 /\*  
427 if temp == 0 then c1 stays the same and c2 is flipped  
428 if temp == 1 then c2 stays the same and c1 is flipped  
429 \*/  
430 if(temp == 0 && c2GreenBinary.charAt(j) == '1')  
431 {  
432 char[] c2Array = c2GreenBinary.toCharArray();  
433 c2Array[j] = '0';  
434 c2GreenBinary = new String(c2Array);  
435 }  
436 else if(temp == 0 && c2GreenBinary.charAt(j) == '0')  
437 {  
438 char[] c2Array = c2GreenBinary.toCharArray();  
439 c2Array[j] = '1';  
440 c2GreenBinary = new String(c2Array);  
441 }  
442 else if(temp == 1 && c1GreenBinary.charAt(j) == '1')  
443 {  
444 char[] c1Array = c1GreenBinary.toCharArray();  
445 c1Array[j] = '0';  
446 c1GreenBinary = new String(c1Array);  
447 }  
448 else  
449 {  
450 char[] c1Array = c1GreenBinary.toCharArray();  
451 c1Array[j] = '1';  
452 c1GreenBinary = new String(c1Array);  
453 }  
454 }  
455 else  
456 {  
457 Random rand = new Random();  
458 int temp = rand.nextInt(20) % 2;  
459 /\*  
460 if temp == 0 then c2 bit is set to c1 bit  
461 if temp == 1 then c1 bit is set to c2 bit  
462 \*/  
463 if(temp == 0)  
464 {  
465 char[] c2Array = c2GreenBinary.toCharArray();  
466 c2Array[j] = c1GreenBinary.charAt(j);  
467 c2GreenBinary = new String(c2Array);  
468 }  
469 else  
470 {  
471 char[] c1Array = c1GreenBinary.toCharArray();  
472 c1Array[j] = c2GreenBinary.charAt(j);  
473 c1GreenBinary = new String(c1Array);  
474 }  
475 }  
476 }  
477   
478 int c1Blue = (cover[0][i] & 0x000000ff);  
479 int c2Blue = (cover[1][i] & 0x000000ff);  
480 int secretBlue = (blue[i] & 0x000000ff);  
481 String c1BlueBinary = String.format("%8s", Integer.toBinaryString(c1Blue)).replace(" ", "0");  
482 String c2BlueBinary = String.format("%8s", Integer.toBinaryString(c2Blue)).replace(" ", "0");  
483 String secretBlueBinary = String.format("%8s", Integer.toBinaryString(secretBlue)).replace(" ", "0");  
484   
485 for(int j = 0; j < secretBlueBinary.length(); j++)  
486 {  
487 if(secretBlueBinary.charAt(j) == '1'  
488 && c1BlueBinary.charAt(j) == c2BlueBinary.charAt(j))  
489 {  
490 Random rand = new Random();  
491 int temp = rand.nextInt(20) % 2;  
492 /\*  
493 if temp == 0 then c1 stays the same and c2 is flipped  
494 if temp == 1 then c2 stays the same and c1 is flipped  
495 \*/  
496 if(temp == 0 && c2BlueBinary.charAt(j) == '1')  
497 {  
498 char[] c2Array = c2BlueBinary.toCharArray();  
499 c2Array[j] = '0';  
500 c2BlueBinary = new String(c2Array);  
501 }  
502 else if(temp == 0 && c2BlueBinary.charAt(j) == '0')  
503 {  
504 char[] c2Array = c2BlueBinary.toCharArray();  
505 c2Array[j] = '1';  
506 c2BlueBinary = new String(c2Array);  
507 }  
508 else if(temp == 1 && c1BlueBinary.charAt(j) == '1')  
509 {  
510 char[] c1Array = c1BlueBinary.toCharArray();  
511 c1Array[j] = '0';  
512 c1BlueBinary = new String(c1Array);  
513 }  
514 else  
515 {  
516 char[] c1Array = c1BlueBinary.toCharArray();  
517 c1Array[j] = '1';  
518 c1BlueBinary = new String(c1Array);  
519 }  
520 }  
521 else  
522 {  
523 Random rand = new Random();  
524 int temp = rand.nextInt(20) % 2;  
525 /\*  
526 if temp == 0 then c2 bit is set to c1 bit  
527 if temp == 1 then c1 bit is set to c2 bit  
528 \*/  
529 if(temp == 0)  
530 {  
531 char[] c2Array = c2BlueBinary.toCharArray();  
532 c2Array[j] = c1BlueBinary.charAt(j);  
533 c2BlueBinary = new String(c2Array);  
534 }  
535 else  
536 {  
537 char[] c1Array = c1BlueBinary.toCharArray();  
538 c1Array[j] = c2BlueBinary.charAt(j);  
539 c1BlueBinary = new String(c1Array);  
540 }  
541 }  
542 }  
543   
544 int row = i / imgWidth;  
545 int column = i % imgWidth;  
546   
547 /\*  
548 //Test for lightening  
549 \*/  
550 Collections.shuffle(colorOrder);  
551   
552 if(colorOrder.get(0).equals("red"))  
553 {  
554 encoded1[2\*row][2\*column] = (Integer.parseInt(c1RedBinary, 2)) << 16;   
555 encoded1[2\*row][2\*column] += (Integer.parseInt("10000000", 2)) << 8;  
556 encoded1[2\*row][2\*column] += (Integer.parseInt("10000000", 2));  
557   
558 encoded2[2\*row][2\*column] = (Integer.parseInt(c2RedBinary, 2)) << 16;   
559 encoded2[2\*row][2\*column] += (Integer.parseInt("10000000", 2)) << 8;  
560 encoded2[2\*row][2\*column] += (Integer.parseInt("10000000", 2));  
561 }  
562 else if(colorOrder.get(0).equals("green"))  
563 {  
564 encoded1[2\*row][2\*column] += (Integer.parseInt("10000000", 2)) << 16;  
565 encoded1[2\*row][2\*column] += (Integer.parseInt(c1GreenBinary, 2)) << 8;  
566 encoded1[2\*row][2\*column] += (Integer.parseInt("10000000", 2));  
567   
568 encoded2[2\*row][2\*column] += (Integer.parseInt("10000000", 2)) << 16;  
569 encoded2[2\*row][2\*column] += (Integer.parseInt(c2GreenBinary, 2)) << 8;  
570 encoded2[2\*row][2\*column] += (Integer.parseInt("10000000", 2));  
571 }  
572 else if(colorOrder.get(0).equals("blue"))  
573 {  
574 encoded1[2\*row][2\*column] += (Integer.parseInt("10000000", 2)) << 16;  
575 encoded1[2\*row][2\*column] += (Integer.parseInt("10000000", 2)) << 8;  
576 encoded1[2\*row][2\*column] += (Integer.parseInt(c1BlueBinary, 2));  
577   
578 encoded2[2\*row][2\*column] += (Integer.parseInt("10000000", 2)) << 16;  
579 encoded2[2\*row][2\*column] += (Integer.parseInt("10000000", 2)) << 8;  
580 encoded2[2\*row][2\*column] += (Integer.parseInt(c2BlueBinary, 2));  
581 }  
582 else  
583 {  
584 encoded1[2\*row][2\*column] = cover1[row][column];  
585 encoded2[2\*row][2\*column] = cover2[row][column];  
586 }  
587   
588 if(colorOrder.get(1).equals("red"))  
589 {  
590 encoded1[2\*row][2\*column + 1] = (Integer.parseInt(c1RedBinary, 2)) << 16;   
591 encoded1[2\*row][2\*column + 1] += (Integer.parseInt("10000000", 2)) << 8;  
592 encoded1[2\*row][2\*column + 1] += (Integer.parseInt("10000000", 2));  
593   
594 encoded2[2\*row][2\*column + 1] = (Integer.parseInt(c2RedBinary, 2)) << 16;   
595 encoded2[2\*row][2\*column + 1] += (Integer.parseInt("10000000", 2)) << 8;  
596 encoded2[2\*row][2\*column + 1] += (Integer.parseInt("10000000", 2));  
597 }  
598 else if(colorOrder.get(1).equals("green"))  
599 {  
600 encoded1[2\*row][2\*column + 1] += (Integer.parseInt("10000000", 2)) << 16;  
601 encoded1[2\*row][2\*column + 1] += (Integer.parseInt(c1GreenBinary, 2)) << 8;  
602 encoded1[2\*row][2\*column + 1] += (Integer.parseInt("10000000", 2));  
603   
604 encoded2[2\*row][2\*column + 1] += (Integer.parseInt("10000000", 2)) << 16;  
605 encoded2[2\*row][2\*column + 1] += (Integer.parseInt(c2GreenBinary, 2)) << 8;  
606 encoded2[2\*row][2\*column + 1] += (Integer.parseInt("10000000", 2));  
607 }  
608 else if(colorOrder.get(1).equals("blue"))  
609 {  
610 encoded1[2\*row][2\*column + 1] += (Integer.parseInt("10000000", 2)) << 16;  
611 encoded1[2\*row][2\*column + 1] += (Integer.parseInt("10000000", 2)) << 8;  
612 encoded1[2\*row][2\*column + 1] += (Integer.parseInt(c1BlueBinary, 2));  
613   
614 encoded2[2\*row][2\*column + 1] += (Integer.parseInt("10000000", 2)) << 16;  
615 encoded2[2\*row][2\*column + 1] += (Integer.parseInt("10000000", 2)) << 8;  
616 encoded2[2\*row][2\*column + 1] += (Integer.parseInt(c2BlueBinary, 2));  
617 }  
618 else  
619 {  
620 encoded1[2\*row][2\*column + 1] = cover1[row][column];  
621 encoded2[2\*row][2\*column + 1] = cover2[row][column];  
622 }  
623   
624 if(colorOrder.get(2).equals("red"))  
625 {  
626 encoded1[2\*row + 1][2\*column] = (Integer.parseInt(c1RedBinary, 2)) << 16;   
627 encoded1[2\*row + 1][2\*column] += (Integer.parseInt("10000000", 2)) << 8;  
628 encoded1[2\*row + 1][2\*column] += (Integer.parseInt("10000000", 2));  
629   
630 encoded2[2\*row + 1][2\*column] = (Integer.parseInt(c2RedBinary, 2)) << 16;   
631 encoded2[2\*row + 1][2\*column] += (Integer.parseInt("10000000", 2)) << 8;  
632 encoded2[2\*row + 1][2\*column] += (Integer.parseInt("10000000", 2));  
633 }  
634 else if(colorOrder.get(2).equals("green"))  
635 {  
636 encoded1[2\*row + 1][2\*column] += (Integer.parseInt("10000000", 2)) << 16;  
637 encoded1[2\*row + 1][2\*column] += (Integer.parseInt(c1GreenBinary, 2)) << 8;  
638 encoded1[2\*row + 1][2\*column] += (Integer.parseInt("10000000", 2));  
639   
640 encoded2[2\*row + 1][2\*column] += (Integer.parseInt("10000000", 2)) << 16;  
641 encoded2[2\*row + 1][2\*column] += (Integer.parseInt(c2GreenBinary, 2)) << 8;  
642 encoded2[2\*row + 1][2\*column] += (Integer.parseInt("10000000", 2));  
643 }  
644 else if(colorOrder.get(2).equals("blue"))  
645 {  
646 encoded1[2\*row + 1][2\*column] += (Integer.parseInt("10000000", 2)) << 16;  
647 encoded1[2\*row + 1][2\*column] += (Integer.parseInt("10000000", 2)) << 8;  
648 encoded1[2\*row + 1][2\*column] += (Integer.parseInt(c1BlueBinary, 2));  
649   
650 encoded2[2\*row + 1][2\*column] += (Integer.parseInt("10000000", 2)) << 16;  
651 encoded2[2\*row + 1][2\*column] += (Integer.parseInt("10000000", 2)) << 8;  
652 encoded2[2\*row + 1][2\*column] += (Integer.parseInt(c2BlueBinary, 2));  
653 }  
654 else  
655 {  
656 encoded1[2\*row + 1][2\*column] = cover1[row][column];  
657 encoded2[2\*row + 1][2\*column] = cover2[row][column];  
658 }  
659   
660 if(colorOrder.get(3).equals("red"))  
661 {  
662 encoded1[2\*row + 1][2\*column + 1] = (Integer.parseInt(c1RedBinary, 2)) << 16;   
663 encoded1[2\*row + 1][2\*column + 1] += (Integer.parseInt("10000000", 2)) << 8;  
664 encoded1[2\*row + 1][2\*column + 1] += (Integer.parseInt("10000000", 2));  
665   
666 encoded2[2\*row + 1][2\*column + 1] = (Integer.parseInt(c2RedBinary, 2)) << 16;   
667 encoded2[2\*row + 1][2\*column + 1] += (Integer.parseInt("10000000", 2)) << 8;  
668 encoded2[2\*row + 1][2\*column + 1] += (Integer.parseInt("10000000", 2));  
669 }  
670 else if(colorOrder.get(3).equals("green"))  
671 {  
672 encoded1[2\*row + 1][2\*column + 1] += (Integer.parseInt("10000000", 2)) << 16;  
673 encoded1[2\*row + 1][2\*column + 1] += (Integer.parseInt(c1GreenBinary, 2)) << 8;  
674 encoded1[2\*row + 1][2\*column + 1] += (Integer.parseInt("10000000", 2));  
675   
676 encoded2[2\*row + 1][2\*column + 1] += (Integer.parseInt("10000000", 2)) << 16;  
677 encoded2[2\*row + 1][2\*column + 1] += (Integer.parseInt(c2GreenBinary, 2)) << 8;  
678 encoded2[2\*row + 1][2\*column + 1] += (Integer.parseInt("10000000", 2));  
679 }  
680 else if(colorOrder.get(3).equals("blue"))  
681 {  
682 encoded1[2\*row + 1][2\*column + 1] += (Integer.parseInt("10000000", 2)) << 16;  
683 encoded1[2\*row + 1][2\*column + 1] += (Integer.parseInt("10000000", 2)) << 8;  
684 encoded1[2\*row + 1][2\*column + 1] += (Integer.parseInt(c1BlueBinary, 2));  
685   
686 encoded2[2\*row + 1][2\*column + 1] += (Integer.parseInt("10000000", 2)) << 16;  
687 encoded2[2\*row + 1][2\*column + 1] += (Integer.parseInt("10000000", 2)) << 8;  
688 encoded2[2\*row + 1][2\*column + 1] += (Integer.parseInt(c2BlueBinary, 2));  
689 }  
690 else  
691 {  
692 encoded1[2\*row + 1][2\*column + 1] = cover1[row][column];  
693 encoded2[2\*row + 1][2\*column + 1] = cover2[row][column];  
694 }  
695 //\*/  
696 }  
697   
698 n = 0;  
699 imgHeight \*= 2;  
700 imgWidth \*= 2;  
701 encryptedShareRGB = new int[2][imgHeight \* imgWidth];  
702 for(int k = 0; k < imgHeight; k++)  
703 {  
704 for(int j = 0; j < imgWidth; j++)  
705 {  
706 encryptedShareRGB[0][n] = encoded1[k][j];  
707 encryptedShareRGB[1][n] = encoded2[k][j];  
708 n += 1;  
709 }  
710 }  
711 }  
712   
713 /\*\*  
714 \* Method TBD.  
715 \* Want to focus on decrypting images and not worrying about if this would  
716 \* work on transparencies.  
717 \*/  
718 public void decryptImage()  
719 {  
720 //Make a 2d array of pixel arrays  
721 int[][] embeddedPixels = new int[numSharesToDecrypt][imgWidth \* imgHeight];  
722 secretMsgPixels = new int[imgWidth \* imgHeight];  
723   
724 //getRGB pixels of BufferedImages  
725 for(int i = 0; i < numSharesToDecrypt; i++)  
726 {  
727 sharesToDecrypt[i].getRGB(0, 0, imgWidth, imgHeight, embeddedPixels[i], 0, imgWidth);  
728 }  
729   
730 int numOfPixels = embeddedPixels[0].length;  
731 for(int i = 0; i < numOfPixels; i++)  
732 {  
733   
734 int redVal1 = (embeddedPixels[0][i] & 0x00ff0000) >> 16;  
735 int greenVal1 = (embeddedPixels[0][i] & 0x0000ff00) >> 8;  
736 int blueVal1 = (embeddedPixels[0][i] & 0x000000ff);  
737   
738 int redVal2 = (embeddedPixels[1][i] & 0x00ff0000) >> 16;  
739 int greenVal2 = (embeddedPixels[1][i] & 0x0000ff00) >> 8;  
740 int blueVal2 = (embeddedPixels[1][i] & 0x000000ff);  
741   
742 //Need to XOR the color concentrations  
743 //XORing mimics stacking transparencies  
744 int redVal = (int)(redVal1 ^ redVal2);  
745 int greenVal = (int)(greenVal1 ^ greenVal2);  
746 int blueVal = (int)(blueVal1 ^ blueVal2);  
747   
748 Color decryptedColor = new Color(redVal, greenVal, blueVal);  
749 secretMsgPixels[i] = decryptedColor.getRGB();  
750 }  
751 }  
752   
753 /\*\*  
754 \* Method decrypts two encoded images by XOR-ing the binary color values together.  
755 \* The XOR technique decrypts the encoded images as if they were printed on  
756 \* transparencies and physically stacked.  
757 \*/  
758 public void decryptImageTransparencyMethod()  
759 {  
760   
761 //Make a 2d array of pixel arrays  
762 int[][] embeddedPixels = new int[numSharesToDecrypt][imgWidth \* imgHeight];  
763   
764 //getRGB pixels of BufferedImages  
765 for(int i = 0; i < numSharesToDecrypt; i++)  
766 {  
767 sharesToDecrypt[i].getRGB(0, 0, imgWidth, imgHeight, embeddedPixels[i], 0, imgWidth);  
768 }  
769   
770 secretMsgPixels = new int[(imgWidth / 2) \* (imgHeight / 2)];  
771 int[][] secretImg = new int[imgHeight / 2][imgWidth / 2];  
772 int[][] encoded1 = new int[imgHeight][imgWidth];  
773 int[][] encoded2 = new int[imgHeight][imgWidth];  
774   
775 int n = 0;  
776 for(int i = 0; i < imgHeight; i++)  
777 {  
778 for(int j = 0; j < imgWidth; j++)  
779 {  
780 encoded1[i][j] = embeddedPixels[0][n];  
781 encoded2[i][j] = embeddedPixels[1][n];  
782 n++;  
783 }  
784 }  
785   
786 imgHeight = imgHeight / 2;  
787 imgWidth = imgWidth / 2;  
788 for(int r = 0; r < imgHeight; r++)  
789 {  
790 for(int c = 0; c < imgWidth; c++)  
791 {  
792 int redConcentrationAvg = 0;  
793 int greenConcentrationAvg = 0;  
794 int blueConcentrationAvg = 0;  
795   
796 redConcentrationAvg += (encoded1[2 \* r][2 \* c] & 0x00ff0000) >> 16;  
797 redConcentrationAvg += (encoded1[2 \* r][2 \* c + 1] & 0x00ff0000) >> 16;  
798 redConcentrationAvg += (encoded1[2 \* r + 1][2 \* c] & 0x00ff0000) >> 16;  
799 redConcentrationAvg += (encoded1[2 \* r + 1][2 \* c + 1] & 0x00ff0000) >> 16;  
800   
801 redConcentrationAvg += (encoded2[2 \* r][2 \* c] & 0x00ff0000) >> 16;  
802 redConcentrationAvg += (encoded2[2 \* r][2 \* c + 1] & 0x00ff0000) >> 16;  
803 redConcentrationAvg += (encoded2[2 \* r + 1][2 \* c] & 0x00ff0000) >> 16;  
804 redConcentrationAvg += (encoded2[2 \* r + 1][2 \* c + 1] & 0x00ff0000) >> 16;  
805   
806 greenConcentrationAvg += (encoded1[2 \* r][2 \* c] & 0x0000ff00) >> 8;  
807 greenConcentrationAvg += (encoded1[2 \* r][2 \* c + 1] & 0x0000ff00) >> 8;  
808 greenConcentrationAvg += (encoded1[2 \* r + 1][2 \* c] & 0x0000ff00) >> 8;  
809 greenConcentrationAvg += (encoded1[2 \* r + 1][2 \* c + 1] & 0x0000ff00) >> 8;  
810   
811 greenConcentrationAvg += (encoded2[2 \* r][2 \* c] & 0x0000ff00) >> 8;  
812 greenConcentrationAvg += (encoded2[2 \* r][2 \* c + 1] & 0x0000ff00) >> 8;  
813 greenConcentrationAvg += (encoded2[2 \* r + 1][2 \* c] & 0x0000ff00) >> 8;  
814 greenConcentrationAvg += (encoded2[2 \* r + 1][2 \* c + 1] & 0x0000ff00) >> 8;  
815   
816 blueConcentrationAvg += (encoded1[2 \* r][2 \* c] & 0x000000ff);  
817 blueConcentrationAvg += (encoded1[2 \* r][2 \* c + 1] & 0x000000ff);  
818 blueConcentrationAvg += (encoded1[2 \* r + 1][2 \* c] & 0x000000ff);  
819 blueConcentrationAvg += (encoded1[2 \* r + 1][2 \* c + 1] & 0x000000ff);  
820   
821 blueConcentrationAvg += (encoded2[2 \* r][2 \* c] & 0x000000ff) >> 16;  
822 blueConcentrationAvg += (encoded2[2 \* r][2 \* c + 1] & 0x000000ff) >> 16;  
823 blueConcentrationAvg += (encoded2[2 \* r + 1][2 \* c] & 0x000000ff) >> 16;  
824 blueConcentrationAvg += (encoded2[2 \* r + 1][2 \* c + 1] & 0x000000ff) >> 16;  
825   
826 redConcentrationAvg /= 8;  
827 greenConcentrationAvg /= 8;  
828 blueConcentrationAvg /= 8;  
829   
830 Color decryptedColor = new Color(redConcentrationAvg, greenConcentrationAvg, blueConcentrationAvg);  
831 secretImg[r][c] = decryptedColor.getRGB();  
832 }  
833 }  
834   
835 int secretIndex = 0;  
836 for(int i = 0; i < imgHeight; i++)  
837 {  
838 for(int j = 0; j < imgWidth; j++)  
839 {  
840 secretMsgPixels[secretIndex] = secretImg[i][j];  
841 secretIndex += 1;  
842 }  
843 }  
844   
845 errorDiffusion(secretMsgPixels);  
846   
847 }  
848   
849 }  
850

Pixel.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 Masters\_Proj;  
 7 import java.awt.Color;  
 8   
 9 /\*\*  
10 \*  
11 \* @author allisonholt  
12 \*/  
13 public class Pixel   
14 {  
15   
16 private int redVal;  
17 private int greenVal;  
18 private int blueVal;  
19   
20 public Pixel(int redIn, int greenIn, int blueIn)  
21 {  
22 redVal = redIn;  
23 greenVal = greenIn;  
24 blueVal = blueIn;  
25 }  
26   
27 //Used to determine if pixel is closer to white than black  
28 public int getConcentration(char color)  
29 {  
30 if(color == 'r')  
31 return redVal;  
32 else if(color == 'g')  
33 return greenVal;  
34 else  
35 return blueVal;  
36 }  
37   
38 }  
39