

Al-imam Mohammad Ibn Saud
Islamic University
Collage of Computer
and Information Science



XSecure

Student name:

Aljuhara Alsadoun - 440023448
Aliah Alhameed - 440020584
Rawan Alshalawi - 440018784
Sara Almashharawi - 440028560

Instructor name:

Dr. Tahani Albalawi

Section: 372

Submission Date:

8/5/2021

Table of Contents

1.0 Overview	1
2.0 Implementation Steps	4
3.0 Code	5
3.1 LoginJFrame	5
3.2 SelectJFrame class	7
3.3 EncryptJFrame class	8
3.4 DecryptJFrame class	10
3.5 RSA class	13
3.6 AES class	13
3.7 userInfo class	14
4.0 Challenges	15
5.0 References	16

1.0 Overview:

In this project we create XSecure application for sharing text in secure way and send it through the network. We use AES to implement encrypt/decrypt, and the application uses symmetric and asymmetric key and encrypt/decrypt the keys using RSA. The system well encrypts the text using AES algorithm with alice public key, and decrypt the text using AES algorithm with alice private key (Figure1).

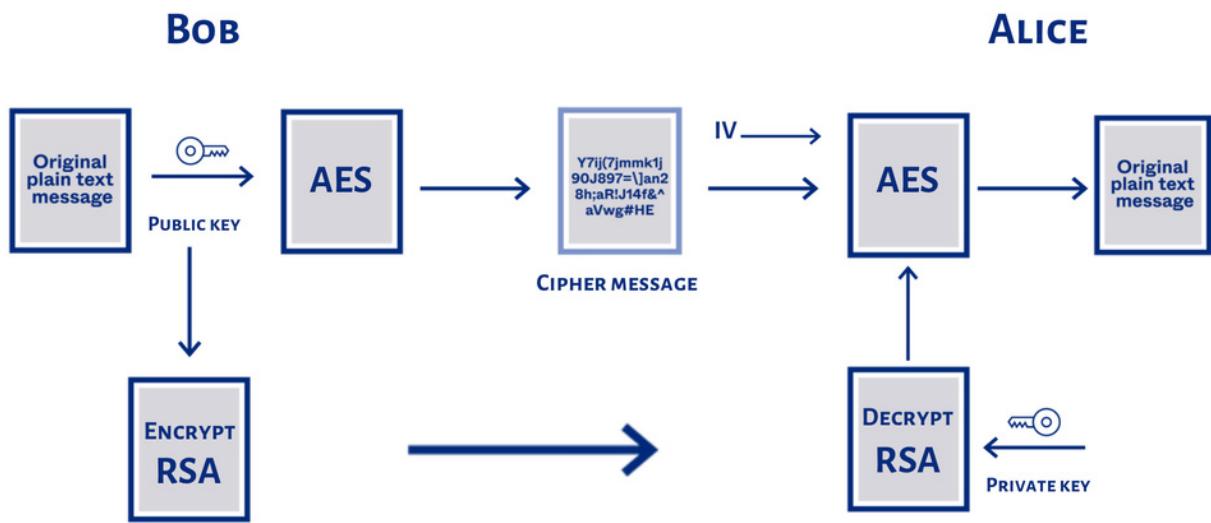


Figure 1

In this project we use java programming language. The GUI of our application contains four windows, the first window (Figure2) able the users to sign in and sign-up using username and password, the system well generates pairs of key, public key, and private key for each user and save the password after applying sha-512, after the user Enter the System window (Figure3) able the user selects the crypto type. In the encrypt window (Figure4) the sender well enters the plain text and select the public key for receiver then the system will encrypt the text using AES, and encrypt the AES key using RSA with the public key, then send the cipher text and encrypted AES key to the receiver , the receiver well receives the cipher text in the decrypt window (Figure5) then selects the private key to decrypt AES key using RSA with his private key ,then decrypt the text using AES.

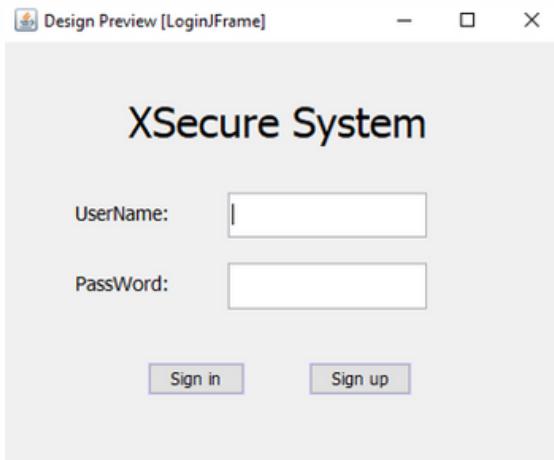


Figure 2

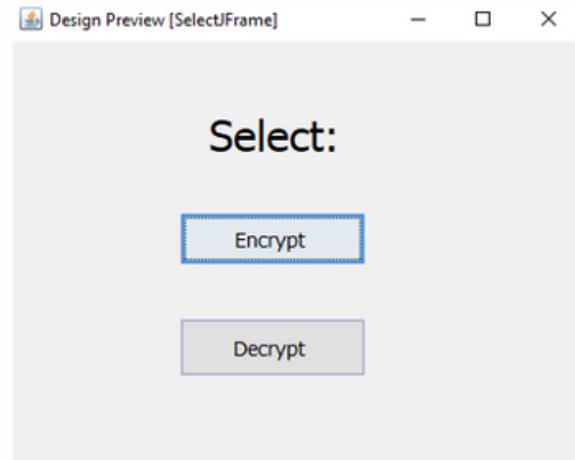


Figure 3

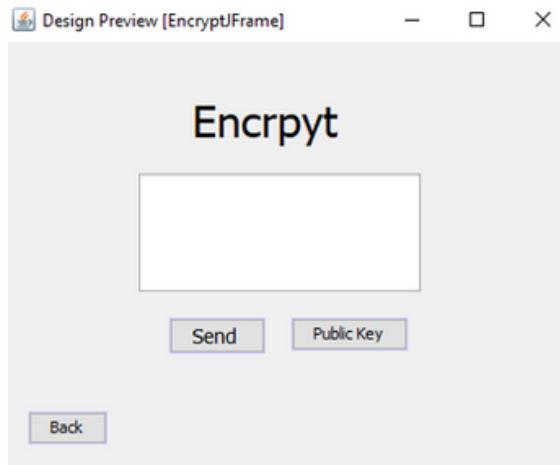


Figure 4

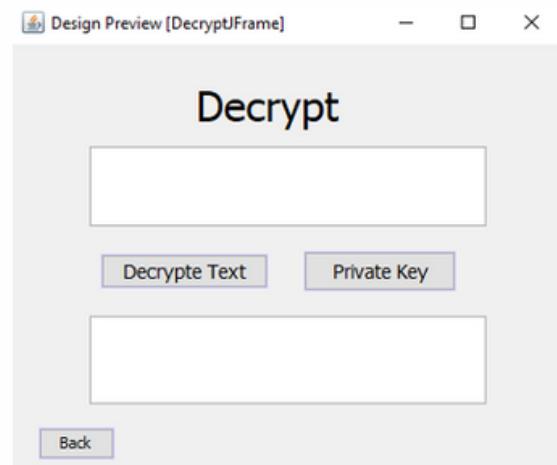


Figure 5

2.0 Implementation Steps:

Login window :

1- Log in the user by select sign in or sign up:

- If a new sign up is selected :

The username and password will be passed through method userInfo.

The username will be registered directly. The password will be hashed in order to keep it encrypted by passing it to method generatHash. This method will receive the password and will hash it using the SHA-512 algorithm and will return it as String to method userInfo After the password is securely saved

Two public and private keys will be generated for each new user that has used KeyPairGeneratorin.java.security

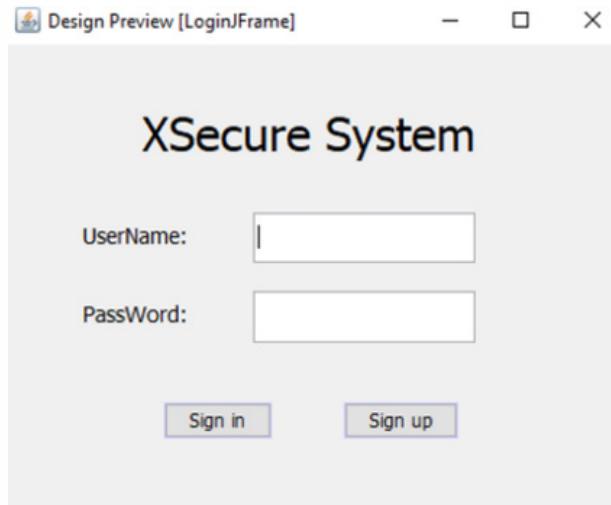
It is an engine class capable of generating a private key and its associated public key using the RSA algorithm. After the two keys are generated, the two keys are securely stored in the file. The public key is saved using the java.security.spec.X509EncodedKeySpec library.

By using the X509EncodedKeySpec object, the getEncoded() method returns the X.509 encoded key, after which the username and public key are saved to the file using FileOutputStream.

As for the private key, it will be saved in the same way as the public key, but the library java.security.spec.PKCS8EncodedKeySpec will be used. The getEncoded() method returns an encryption for the private key

Thus, the two keys are kept securely.

- If a login is selected, it will go directly to the encryption or decryption selection window.



Select window :

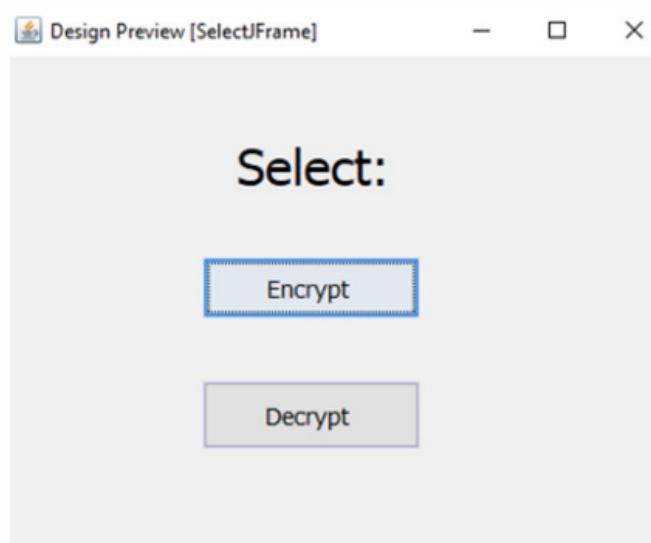
When a selection window, there are two options:

Encryption option:

Encryption field that uses the javax.swing.JFrame library and an object of class EncryptJFrame will be pushed.

Decryption option:

The decryption field that uses the javax.swing.JFrame library and an object of class DecryptJFrame will be pushed.



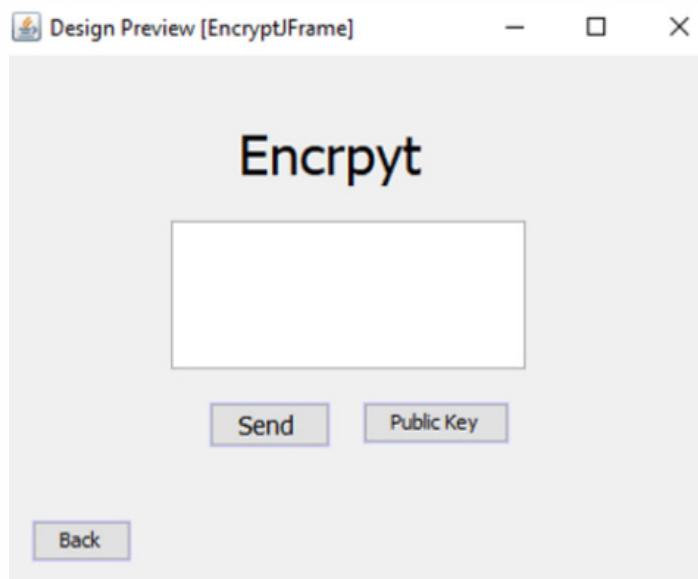
Encrypt window :

If the Encrypt button is selected:

- The plainText will be entered into the text field (jTextField1) of type String

- The public key button must be selected. An encrypted key will be generated to share between the sender and the receiver. The public key button will be selected and read and an AES key generated to generate a secure key. We used Keygenerator, which depends on the correctly classified encrypted random number generator and saved it in the key variable after saving the key. The public key will be converted from String to publickey We used the library javax.swing.filechooser.FileSystemView to save the public key file path in the variable path using the JFileChooser object and then the public key will be converted from String to publickey reading the byte of the file and then encrypting it with X509EncodedKeySpec after that convert byte To a public key using KeyFactory it returns a Key Factory object that converts public/private keys for RSA algorithm and using the generatePublic method. Then the key will be encrypted using RSA algorithm by creating an RSA variable and sending it to the Base64.getEncoder().encodeToString().
(key.getEncoded()), pubkey)Encoder using getEncoder() then get the encoded string by passing the private byte value Enter the actual string in the encodeToString() method to convert it to text. Thus, the public key was selected for the future and an encrypted key was generated using the RSA algorithm.

- After entering the text and choosing the public key, when you press the encryption button, the text will be encrypted and displayed encrypted. At first, an AES variable was created and then sent to method encrypt with the text and the encrypted key, and when you receive it, you convert the text to a byte and then choose encryption mode Then initialize the encryption using toCipher.init with key and parameters then store the IV as a string to share over socket IV and then start encrypting the text and saving it in byte using toCipher. byte to string by sending it to method encode which converts byte to string using Base64.getEncoder().encodeToString() After encoding the text we save the iv and then send it to the receiver for decoding.

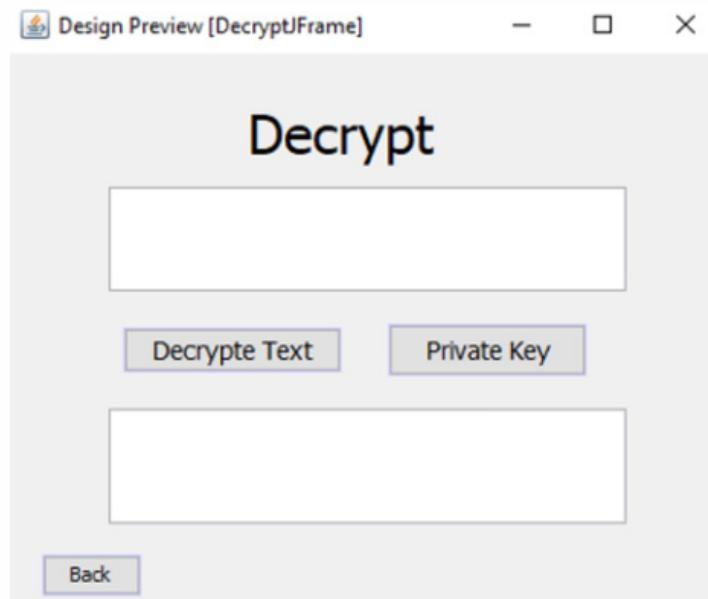


Decrypt window :

If decrypt is selected:

This button receives an encrypted text and decrypts it. Initially, an RSA object will be created to decrypt the symmetric key, then store the secret key in a byte string to convert it from String to SecretKey byte. The rsa variable with key and private key is sent to method decrypt (this method decrypts). Encrypting the ciphertext using the private key The key is sent to method encode and the string is converted to a byte using Base64.getDecoder().decode()

Then the encrypted text is converted using Cipher.getInstance(algorithm) It decrypts the text using the private key and then initializes the decryption using toPlain.init with the key and parameters and then starts to decrypt the text and saves it in a byte using toPlain.doFinal it decrypts in the process From one part depending on how this encryption is configured then it is returned byte) Then the key will be re-created using secretKeySpec We used secretkey which is an interface that defines it as a secret (symmetric) key through which the decryption is done, and then start the AES object to decrypt The file using SecretKey by creating an AES variable and then sending it to method decryp with the encrypted text and secretKey and iv when decrypted the original text will be returned as a string after which the decrypted text is displayed in PlainTextField and thus the decryption process is done.



3.0 Code:

LoginJFrame :

```

93         .addGroup(layout.createSequentialGroup()
94             .addGap(103, 103)
95             .addComponent(jButton2)
96             .addGap(46, 46)
97             .addComponent(jButton3))
98         .addGroup(layout.createSequentialGroup()
99             .addGap(184, 184)
100             .addComponent(jLabel14)))
101     .addContainerGap(98, Short.MAX_VALUE))
102 );
103 layout.setVerticalGroup(
104     layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
105     .addGroup(layout.createSequentialGroup()
106         .addGap(38, 38, 38)
107         .addComponent(jLabel1)
108         .addGap(32, 32, 32)
109         .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
110             .addComponent(jTextField1, javax.swing.GroupLayout.PREFERRED_SIZE, 32, javax.swing.GroupLayout.PREFERRED_SIZE)
111             .addComponent(jLabel2))
112         .addGap(18, 18, 18)
113         .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
114             .addComponent(jLabel3)
115             .addComponent(jPasswordField1, javax.swing.GroupLayout.PREFERRED_SIZE, 33, javax.swing.GroupLayout.PREFERRED_SIZE))
116         .addGap(28, 28, 28)
117         .addComponent(jLabel4)
118         .addGap(18, 18, 18)
119         .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
120             .addComponent(jButton2)
121             .addComponent(jButton2))
122         .addContainerGap(49, Short.MAX_VALUE))
123 );
124
125     pack();
126 } // </editor-fold>
127 // method to signin users
128 private void SignInButton(java.awt.event.ActionEvent evt) {
129     try {
130         String Pass = jPasswordField1.getText();
131         String User = jTextField1.getText();
132         // to compare intered password and hash it with user password
133         for(userInfo s : users)
134             if (User.equals(s.userName)&&s.generateHash(Pass).equals(s.password)){
135                 SelectJFrame select = new SelectJFrame();
136                 select.show();
137                 dispose();
138
139                 jLabel4.setText("incorrect password or username");
140
141             } catch (Exception ex) {
142                 System.out.print(ex);
143             }
144     }
145     // method to sign up new user and add user to users array
146     private void SignUpButton(java.awt.event.ActionEvent evt) throws InterruptedException {
147
148         String Pass = jPasswordField1.getText();
149         String User = jTextField1.getText();
150
151         try {
152             userInfo newUser = new userInfo(User,Pass);
153             users.add(newUser);
154         }
155         catch (Exception ex) {
156             System.out.print(ex);
157         }
158         jLabel4.setText("sign up successfully");
159         SelectJFrame select = new SelectJFrame();
160         select.show();
161         dispose();
162     }
163     public static void main(String args[]) {
164
165
166
167     try {
168
169         userInfo admin = new userInfo("admin","1234");
170         userInfo admin2 = new userInfo("admin2","12345");
171         users.add(admin);
172         users.add(admin2);
173
174         for (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.getInstalledLookAndFeels()) {
175             if ("Nimbus".equals(info.getName())){
176                 javax.swing.UIManager.setLookAndFeel(info.getClassName());
177                 break;
178             }
179         }
180     } catch (ClassNotFoundException ex) {
181         java.util.logging.Logger.getLogger(LoginJFrame.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
182     } catch (InstantiationException ex) {
183         java.util.logging.Logger.getLogger(LoginJFrame.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
184     } catch (IllegalAccessException ex) {
185         java.util.logging.Logger.getLogger(LoginJFrame.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
186     } catch (java.awt.UnsupportedLookAndFeelException ex) {
187         java.util.logging.Logger.getLogger(LoginJFrame.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
188     } catch (Exception ex){
189         System.out.println("No User");
190     } //</editor-fold>
191     /* Create and display the form */
192     java.awt.EventQueue.invokeLater(new Runnable() {
193         public void run() {
194             new LoginJFrame().setVisible(true);
195         }
196     });
197 }
198 // Variables declaration - do not modify
199 private javax.swing.JButton jButton2;
200 private javax.swing.JButton jButton3;
201 private javax.swing.JLabel jLabel1;
202 private javax.swing.JLabel jLabel2;
203 private javax.swing.JLabel jLabel3;
204 private javax.swing.JLabel jLabel14;
205 private javax.swing.JPasswordField jPasswordField1;
206 private javax.swing.JTextField jTextField1;
207 // End of variables declaration

```

SelectJFrame class :

```
17 public class SelectJFrame extends javax.swing.JFrame {
18
19     // Creates new form SelectJFrame
20     public SelectJFrame() {
21         initComponents();
22     }
23
24     /**
25      * This method is called from within the constructor to initialize the form.
26      * WARNING: Do NOT modify this code. The content of this method is always
27      * regenerated by the Form Editor.
28     */
29     @SuppressWarnings("unchecked")
30     // <editor-fold defaultstate="collapsed" desc="Generated Code">
31     private void initComponents() {
32
33         jLabel1 = new javax.swing.JLabel();
34         jButton1 = new javax.swing.JButton();
35         jButton2 = new javax.swing.JButton();
36
37         setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
38
39         jLabel1.setFont(new java.awt.Font("Tahoma", 0, 30)); // NOI18N
40         jLabel1.setText("Select File:");
41
42         jButton1.setBackground(new java.awt.Color(204, 204, 255));
43         jButton1.setFont(new java.awt.Font("Tahoma", 0, 14)); // NOI18N
44         jButton1.setText("Encrypt File");
45         jButton1.addActionListener(new java.awt.event.ActionListener() {
46             public void actionPerformed(java.awt.event.ActionEvent evt) {
47                 jButton1ActionPerformed(evt);
48             }
49         });
50
51         jButton2.setBackground(new java.awt.Color(204, 204, 255));
52         jButton2.setFont(new java.awt.Font("Tahoma", 0, 14)); // NOI18N
53         jButton2.setText("Decrypt File");
54         jButton2.addActionListener(new java.awt.event.ActionListener() {
55             public void actionPerformed(java.awt.event.ActionEvent evt) {
56                 jButton2ActionPerformed(evt);
57             }
58         });
59
60         javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
61         getContentPane().setLayout(layout);
62         layout.setHorizontalGroup(
63             layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
64                 .addGroup(layout.createSequentialGroup()
65                     .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
66                         .addGroup(layout.createSequentialGroup()
67                             .addComponent(jLabel1)
68                         .addGap(18, 18, 18)
69                         .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
70                             .addGroup(layout.createSequentialGroup()
71                                 .addComponent(jButton1, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
72                                 .addComponent(jButton2, javax.swing.GroupLayout.DEFAULT_SIZE, 129, Short.MAX_VALUE))
73                             .addGap(143, 143, 143)))
74                     .addGap(18, 18, 18)
75                     .addComponent(jLabel1)
76                     .addGap(18, 18, 18)
77                     .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
78                         .addGroup(layout.createSequentialGroup()
79                             .addComponent(jButton1, javax.swing.GroupLayout.PREFERRED_SIZE, 36, javax.swing.GroupLayout.PREFERRED_SIZE)
80                             .addComponent(jButton2, javax.swing.GroupLayout.PREFERRED_SIZE, 40, javax.swing.GroupLayout.PREFERRED_SIZE))
81                         .addGap(66, 66, 66)))
82                 .addGap(49, 49, 49)
83                 .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
84                     .addGroup(layout.createSequentialGroup()
85                         .addComponent(jButton1, javax.swing.GroupLayout.PREFERRED_SIZE, 36, javax.swing.GroupLayout.PREFERRED_SIZE)
86                         .addComponent(jButton2, javax.swing.GroupLayout.PREFERRED_SIZE, 40, javax.swing.GroupLayout.PREFERRED_SIZE))
87                     .addGap(66, 66, 66)))
88             .addGap(38, 38, 38)
89             .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
90                 .addComponent(jButton1, javax.swing.GroupLayout.PREFERRED_SIZE, 36, javax.swing.GroupLayout.PREFERRED_SIZE)
91                 .addComponent(jButton2, javax.swing.GroupLayout.PREFERRED_SIZE, 40, javax.swing.GroupLayout.PREFERRED_SIZE))
92             .addGap(66, 66, 66)
93             .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
94                 .addComponent(jButton1, javax.swing.GroupLayout.PREFERRED_SIZE, 36, javax.swing.GroupLayout.PREFERRED_SIZE)
95                 .addComponent(jButton2, javax.swing.GroupLayout.PREFERRED_SIZE, 40, javax.swing.GroupLayout.PREFERRED_SIZE))
96             .addGap(66, 66, 66)
97             .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
98                 .addComponent(jButton1, javax.swing.GroupLayout.PREFERRED_SIZE, 36, javax.swing.GroupLayout.PREFERRED_SIZE)
99                 .addComponent(jButton2, javax.swing.GroupLayout.PREFERRED_SIZE))
100            .addGap(66, 66, 66)
101            .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
102                .addComponent(jButton1, javax.swing.GroupLayout.PREFERRED_SIZE, 36, javax.swing.GroupLayout.PREFERRED_SIZE)
103                .addComponent(jButton2, javax.swing.GroupLayout.PREFERRED_SIZE))
104        );
105    }
106
107    pack();
108 } // </editor-fold>
109
110 private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
111     // to open select file page
112     EncryptJFrame encrypt = new EncryptJFrame();
113     encrypt.show();
114
115     // to close current page and open selected page.
116     dispose();
117 }
118
119 private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
120     // to open select file page
121     DecryptJFrame dncrypt = new DecryptJFrame();
122     dncrypt.show();
123 }
```

```
184     //to close current page and open selected page.
185     dispose();
186 }
187
188 /**
189 * @param args the command line arguments
190 */
191 public static void main(String args[]) {
192     /* Set the Nimbus look and feel */
193     //
194     /* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and feel.
195      * For details see http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html
196     */
197     try {
198         for (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.getInstalledLookAndFeels()) {
199             if ("Nimbus".equals(info.getName())) {
200                 javax.swing.UIManager.setLookAndFeel(info.getClassName());
201                 break;
202             }
203         }
204     } catch (ClassNotFoundException ex) {
205         java.util.logging.Logger.getLogger(SelectJFrame.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
206     } catch (InstantiationException ex) {
207         java.util.logging.Logger.getLogger(SelectJFrame.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
208     } catch (IllegalAccessException ex) {
209         java.util.logging.Logger.getLogger(SelectJFrame.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
210     } catch (javax.swing.UnsupportedLookAndFeelException ex) {
211         java.util.logging.Logger.getLogger(SelectJFrame.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
212     }
213     //
214
215     /* Create and display the form */
216     java.awt.EventQueue.invokeLater(new Runnable() {
217         public void run() {
218             new SelectJFrame().setVisible(true);
219         }
220     });
221 }
222
223 // Variables declaration - do not modify
224 private javax.swing.JButton jButton1;
225 private javax.swing.JButton jButton2;
226 private javax.swing.JLabel jLabel1;
227 // End of variables declaration
228 }
```

EncryptJFrame class :

```
import java.io.*;
12 import java.net.*;
13 import java.net.Socket;
14 import java.nio.file.Files;
15 import java.nio.file.Paths;
16 import java.security.KeyFactory;
17 import java.security.PublicKey;
18 import java.security.spec.X509EncodedKeySpec;
19 import java.util.Base64;
20 import javax.crypto.KeyGenerator;
21 import javax.crypto.SecretKey;
22 import javax.swing.JFileChooser;
23 import javax.swing.filechooser.FileSystemView;
24
25 public class EncryptJFrame extends javax.swing.JFrame {
26     static ServerSocket server;
27     static Socket socket;
28     static DataOutputStream outputtext;
29     static DataOutputStream outputiv;
30     static DataOutputStream outputkey;
31     static String symmetricKey;
32     SecretKey key;
33
34     public EncryptJFrame() {
35         initComponents();
36     }
37
38     @SuppressWarnings("unchecked")
39     // <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-BEGIN:initComponents
40     private void initComponents() {
41
42         jLabel3 = new javax.swing.JLabel();
43         jLabel1 = new javax.swing.JLabel();
44         jButton3 = new javax.swing.JButton();
45         jLabel2 = new javax.swing.JLabel();
46         jButton2 = new javax.swing.JButton();
47         jTextField1 = new javax.swing.JTextField();
48         jButton1 = new javax.swing.JButton();
49
50         jLabel3.setText("jLabel3");
51
52         setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
53
54         jLabel1.setFont(new java.awt.Font("Tahoma", 0, 30)); // NOI18N
55         jLabel1.setText("Encript");
56
57         jButton3.setBackground(new java.awt.Color(204, 204, 255));
58         jButton3.setFont(new java.awt.Font("Tahoma", 0, 14)); // NOI18N
59         jButton3.setText("Send ");
60         jButton3.addActionListener(new java.awt.event.ActionListener() {
61             public void actionPerformed(java.awt.event.ActionEvent evt) {
62                 sendButton(evt);
63             }
64         });
65
66         jButton2.setBackground(new java.awt.Color(204, 204, 255));
67         jButton2.setText("Back");
68         jButton2.addActionListener(new java.awt.event.ActionListener() {
```

```

69         public void actionPerformed(java.awt.event.ActionEvent evt) {
70             jButton2ActionPerformed(evt);
71         });
72     });
73
74     jTextField1.addActionListener(new java.awt.event.ActionListener() {
75         public void actionPerformed(java.awt.event.ActionEvent evt) {
76             jTextField1ActionPerformed(evt);
77         }
78     });
79
80     jButton1.setBackground(new java.awt.Color(204, 204, 255));
81     jButton1.setText("Public Key");
82     jButton1.addActionListener(new java.awt.event.ActionListener() {
83         public void actionPerformed(java.awt.event.ActionEvent evt) {
84             selectPublicKButton(evt);
85         }
86     });
87
88     javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
89     getContentPane().setLayout(layout);
90     layout.setHorizontalGroup(
91         layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
92             .addGroup(layout.createSequentialGroup()
93                 .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
94                     .addGroup(layout.createSequentialGroup()
95                         .addGap(133, 133, 133)
96                         .addComponent(jLabel1))
97                     .addGroup(layout.createSequentialGroup()
98                         .addGap(19, 19, 19)
99                         .addComponent(jButton2)))
100                 .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
101                     .addGroup(layout.createSequentialGroup()
102                         .addGap(96, 96, 96)
103                         .addComponent(jTextField1, javax.swing.GroupLayout.PREFERRED_SIZE, 196, javax.swing.GroupLayout.PREFERRED_SIZE)
104                         .addGap(55, 55, 55)
105                         .addComponent(jLabel2)))
106                     .addGroup(layout.createSequentialGroup()
107                         .addGap(117, 117, 117)
108                         .addComponent(jButton3)
109                         .addGap(18, 18, 18)
110                         .addComponent(jButton1)))
111                 .addGap(53, Short.MAX_VALUE)
112             );
113             layout.setVerticalGroup(
114                 layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
115                     .addGroup(layout.createSequentialGroup()
116                         .addComponent(jLabel1)
117                         .addComponent(jTextField1, javax.swing.GroupLayout.PREFERRED_SIZE, 82, javax.swing.GroupLayout.PREFERRED_SIZE))
118                     .addGroup(layout.createSequentialGroup()
119                         .addGap(91, 91, 91)
120                         .addComponent(jLabel2))
121                     .addGroup(layout.createSequentialGroup()
122                         .addGap(18, 18, 18)
123                         .addComponent(jButton1, javax.swing.GroupLayout.PREFERRED_SIZE, 21, javax.swing.GroupLayout.PREFERRED_SIZE))
124                     .addGap(18, 18, 18)
125                     .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
126                         .addComponent(jButton3)
127                         .addComponent(jButton1))
128                     .addGap(48, 48, 48)
129                     .addComponent(jButton2)
130                     .addGap(21, Short.MAX_VALUE))
131             );
132
133             pack();
134         }/// </editor-fold> //GEN-END:initComponents
135
136         // method to encrypted text then send it and yhe iv to the receiver.
137         private void sendButton(java.awt.event.ActionEvent evt) { //GEN-FIRST:event_jButton3ActionPerformed
138
139             try{
140                 String PlainText = jTextField1.getText();
141                 AES aes = new AES();
142                 String cipher = aes.encrypt(PlainText,key);
143                 String iv= aes.IV;
144                 outputtext.writeUTF(cipher);
145                 outputiv.writeUTF(iv);
146
147             }
148             catch(Exception e){
149                 System.out.println("no text enter"+ e.toString());
150             }
151
152         } //GEN-LAST:event_jButton3ActionPerformed
153         // back button
154         private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) { //GEN-FIRST:event_jButton2ActionPerformed
155
156             SelectJFrame select = new SelectJFrame();
157             select.show();
158             dispose();
159         } //GEN-LAST:event_jButton2ActionPerformed
160
161         private void jTextField1ActionPerformed(java.awt.event.ActionEvent evt) { //GEN-FIRST:event_jTextField1ActionPerformed
162
163         } //GEN-LAST:event_jTextField1ActionPerformed
164
165         // method to select and read public key file and generate Symitric key then encrypt it
166         // then send symetric key to reciever
167         private void selectPublicKButton(java.awt.event.ActionEvent evt) { //GEN-FIRST:event_jButton1ActionPerformed
168
169             try{
170                 //Create assymmetric key AES and Create key generator
171                 KeyGenerator generateKey =KeyGenerator.getInstance("AES");
172                 generateKey.init(128);
173                 key = generateKey.generateKey(); // aes key
174                 // store path of public key file
175                 String path="";
176                 JFileChooser j = new JFileChooser(FileSystemView.getFileSystemView());
177                 int r = j.showOpenDialog(null);
178                 if (r == JFileChooser.APPROVE_OPTION) {
179                     path = j.getSelectedFile().getName();
180                 }
181
182                 // convert public key from String to PublicKey
183                 byte[] keyBytes = Files.readAllBytes(Paths.get(path));

```

```

184     X509EncodedKeySpec spec =new X509EncodedKeySpec(keyBytes);
185     KeyFactory kf = KeyFactory.getInstance("RSA");
186     PublicKey pubkey= kf.generatePublic(spec);
187     System.out.println("public key is "+Base64.getEncoder().encodeToString(pubkey.getEncoded()));
188     // encrypt symmetric key
189     RSA a = new RSA();
190     symmetricKey=a.encrypt(Base64.getEncoder().encodeToString(key.getEncoded()), pubkey);
191     outputkey.writeUTF(symmetricKey);
192 }
193 catch(Exception e){
194     System.err.println("no key file"+e.toString());
195 }
196 }//GEN-LAST:event_jButton1ActionPerformed
197
198 /**
199 * @param args the command line arguments
200 */
201 public static void main(String args[]) throws IOException {
202
203     /* Set the Nimbus look and feel */
204
205     //editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) "
206     /* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and feel.
207     * For details see http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html
208     */
209     try {
210         for (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.getInstalledLookAndFeels()) {
211             if ("Nimbus".equals(info.getName())) {
212                 javax.swing.UIManager.setLookAndFeel(info.getClassName());
213                 break;
214             }
215         }
216     } catch (ClassNotFoundException ex) {
217         java.util.logging.Logger.getLogger(EncryptJFrame.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
218     } catch (InstantiationException ex) {
219         java.util.logging.Logger.getLogger(EncryptJFrame.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
220     } catch (IllegalAccessException ex) {
221         java.util.logging.Logger.getLogger(EncryptJFrame.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
222     } catch (javax.swing.UnsupportedLookAndFeelException ex) {
223         java.util.logging.Logger.getLogger(EncryptJFrame.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
224     }
225     //
226
227     /* Create and display the form */
228     java.awt.EventQueue.invokeLater(new Runnable() {
229         public void run() {
230             new EncryptJFrame().setVisible(true);
231         }
232     });
233
234     try{
235         // create socket to send the cipher text
236         server = new ServerSocket(5000);
237         socket = server.accept();
238         outputtext = new DataOutputStream(socket.getOutputStream());
239         outputiv = new DataOutputStream(socket.getOutputStream());
240         outputkey = new DataOutputStream(socket.getOutputStream());
241     }
242     catch(Exception e){
243         System.err.println("no receiver"+e.toString());
244     }
245
246
247
248
249
250     }
251
252     // Variables declaration - do not modify//GEN-BEGIN:variables
253     private javax.swing.JButton jButton1;
254     private javax.swing.JButton jButton2;
255     private javax.swing.JButton jButton3;
256     private javax.swing.JLabel jLabel1;
257     private javax.swing.JLabel jLabel2;
258     private javax.swing.JLabel jLabel3;
259     private javax.swing.JTextField jTextField1;
260     // End of variables declaration//GEN-END:variables
261
262 }
263

```

DecryptJFrame class :

```
7  import java.io.*;
8  import java.net.Socket;
9  import java.nio.file.Files;
10 import java.nio.file.Paths;
11 import java.security.KeyFactory;
12 import java.security.PrivateKey;
13 import java.security.spec.PKCS8EncodedKeySpec;
14 import java.util.Base64;
15 import javax.crypto.SecretKey;
16 import javax.crypto.spec.SecretKeySpec;
17 import javax.swing.JFileChooser;
18 import javax.swing.filechooser.FileSystemView;
19
20
21 public class DecryptJFrame extends javax.swing.JFrame {
22
23     //static ServerSocket server;
24     static Socket socket;
25     static DataInputStream inputkey;
26     static DataInputStream inputtext;
27     static DataInputStream inputiv;
28     static byte[] iv;
29     static String key;
30     static String CipherText;
31     SecretKey secretKey ;
32     PrivateKey PrivateKey;
33
34
35     public DecryptJFrame() {
36         initComponents();
37     }
38
39     @SuppressWarnings("unchecked")
40     // <editor-fold defaultstate="collapsed" desc="Generated Code">
41     private void initComponents() {
42
43         DecryptLabel = new javax.swing.JLabel();
44         DecryptButton = new javax.swing.JButton();
45         ErrorLabel = new javax.swing.JLabel();
46         BackButton = new javax.swing.JButton();
47         CipherTextField = new javax.swing.JTextField();
48         PlainTextField = new javax.swing.JTextField();
49         prKeyButton = new javax.swing.JButton();
50
51         setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
52
53         DecryptLabel.setFont(new java.awt.Font("Tahoma", 0, 30)); // NOI18N
54         DecryptLabel.setText("Decrypt");
55         DecryptLabel.setPreferredSize(new java.awt.Dimension(101, 37));
56
57         DecryptButton.setBackground(new java.awt.Color(204, 204, 255));
58         DecryptButton.setFont(new java.awt.Font("Tahoma", 0, 14)); // NOI18N
59         DecryptButton.setText("Decrypt Text");
60         DecryptButton.addActionListener(new java.awt.event.ActionListener() {
61             public void actionPerformed(java.awt.event.ActionEvent evt) {
62                 decryptButton(evt);
63             }
64         });
65
66         ErrorLabel.setFont(new java.awt.Font("Tahoma", 0, 12)); // NOI18N
67
68         BackButton.setBackground(new java.awt.Color(204, 204, 255));
69         BackButton.setText("Back");
70         BackButton.addActionListener(new java.awt.event.ActionListener() {
71             public void actionPerformed(java.awt.event.ActionEvent evt) {
72                 BackButton(evt);
73             }
74         });
75
76         CipherTextField.setToolTipText("");
77         CipherTextField.addActionListener(new java.awt.event.ActionListener() {
78             public void actionPerformed(java.awt.event.ActionEvent evt) {
79                 jTextField1ActionPerformed(evt);
80             }
81         });
82
83         PlainTextField.setForeground(new java.awt.Color(153, 153, 153));
84         PlainTextField.addActionListener(new java.awt.event.ActionListener() {
85             public void actionPerformed(java.awt.event.ActionEvent evt) {
86                 jTextField2ActionPerformed(evt);
87             }
88         });
89
90         prKeyButton.setBackground(new java.awt.Color(204, 204, 255));
91         prKeyButton.setFont(new java.awt.Font("Tahoma", 0, 14)); // NOI18N
92         prKeyButton.setText("Private Key");
```

```

93     prKeyButton.addActionListener(new java.awt.event.ActionListener() {
94         public void actionPerformed(java.awt.event.ActionEvent evt) {
95             selectPprivateKButton(evt);
96         }
97     });
98
99     javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
100    getContentPane().setLayout(layout);
101    layout.setHorizontalGroup(
102        layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
103            .addGroup(layout.createSequentialGroup()
104                .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
105                    .addGroup(layout.createSequentialGroup()
106                        .addGap(134, 134, 134)
107                        .addComponent(Decryptable, javax.swing.GroupLayout.PREFERRED_SIZE, 110, javax.swing.GroupLayout.PREFERRED_SIZE))
108                    .addGroup(layout.createSequentialGroup()
109                        .addGap(278, 278, 278)
110                        .addComponent(ErrorLabel))
111                    .addGroup(layout.createSequentialGroup()
112                        .addGap(20, 20, 20)
113                        .addComponent(BackButton))
114                    .addGroup(layout.createSequentialGroup()
115                        .addGap(57, 57, 57)
116                        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
117                            .addComponent(CipherTextField)
118                            .addComponent(PlainTextField, javax.swing.GroupLayout.DEFAULT_SIZE, 287, Short.MAX_VALUE)))
119                    .addGroup(layout.createSequentialGroup()
120                        .addGap(65, 65, 65)
121                        .addGap(26, 26, 26)
122                        .addComponent(prKeyButton, javax.swing.GroupLayout.PREFERRED_SIZE, 110, javax.swing.GroupLayout.PREFERRED_SIZE)))
123                    .addGroup(layout.createSequentialGroup()
124                        .addGap(56, Short.MAX_VALUE))
125                );
126            layout.setVerticalGroup(
127                layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
128                    .addGroup(layout.createSequentialGroup()
129                        .addGap(26, 26, 26)
130                        .addComponent(Decryptable, javax.swing.GroupLayout.PREFERRED_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
131                        .addGap(PreferredGap, javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
132                        .addComponent(CipherTextField, javax.swing.GroupLayout.PREFERRED_SIZE, 58, javax.swing.GroupLayout.PREFERRED_SIZE)
133                        .addGap(18, 18, 18)
134                        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
135                            .addComponent(DecryptButton)
136                            .addComponent(prKeyButton, javax.swing.GroupLayout.PREFERRED_SIZE, 29, javax.swing.GroupLayout.PREFERRED_SIZE))
137                        .addGap(18, 18, 18)
138                        .addComponent(PlainTextField, javax.swing.GroupLayout.PREFERRED_SIZE, 64, javax.swing.GroupLayout.PREFERRED_SIZE)
139                        .addGap(PreferredGap, javax.swing.LayoutStyle.ComponentPlacement.RELATED)
140                        .addComponent(ErrorLabel)
141                        .addGap(PreferredGap, javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
142                        .addComponent(BackButton)
143                        .addGap(ContainerGap, javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
144                );
145            pack();
146        }// </editor-fold>
147 // This button receive cipher text and decrypt it
148 private void decryptButton(java.awt.event.ActionEvent evt) {
149
150     try{
151     // initiate RSA object to decrypt Symetric Key
152     RSA rsa= new RSA();
153     //Store secret key in byte String to conver it from String To SecretKey
154     byte[] decodedKey = Base64.getDecoder().decode(rsa.decrypt(key, PrivateKey));
155     // Rebuild key using SecretKeySpec
156     secretKey = new SecretKeySpec(decodedKey, 0, decodedKey.length, "AES");
157     System.out.println("Symetric key after decrypt "+Base64.getEncoder().encodeToString(secretKey.getEncoded()));
158
159     // initiate AES object to decrypt file using SecretKey
160     AES aes = new AES();
161     String plain =aes.decrypt(CipherText, secretKey,iv);
162     // present decrypted text in PlainTextField
163     PlainTextField.setText(PlainTextField.getText().trim()+"\n"+plain);
164   }
165   catch(Exception error){
166     System.err.println("Error! No Key OR Text "+ error.toString());
167   }
168 }
169
170
171 // method to go to main page
172 private void BackButton(java.awt.event.ActionEvent evt) {
173     SelectJFrame select = new SelectJFrame();
174     select.show();
175
176     dispose();
177 }
178
179
180 private void jTextField1ActionPerformed(java.awt.event.ActionEvent evt) {
181     // this text filed will receive the cipher text from the sender.
182
183 }
184
185 private void jTextField2ActionPerformed(java.awt.event.ActionEvent evt) {
186     // present plain text after decryption
187
188 }
189
190 // method to selectPprivateKButton and decrypt the symetricKey
191 private void selectPprivateKButton(java.awt.event.ActionEvent evt) {
192     try {
193         // to select private key file and read it as String
194         String path="";
195         JFileChooser j = new JFileChooser(FileSystemView.getFileSystemView());
196         int r = j.showOpenDialog(null);
197         if (r == JFileChooser.APPROVE_OPTION) {
198             path = j.getSelectedFile().getName();
199             //Store private key in byte String to convert it from String To PrivateKey
200             byte[] keybytes2 = Files.readAllBytes(paths.get(path));
201             PKCS8EncodedKeySpec spec2 =new PKCS8EncodedKeySpec(keybytes2);
202             KeyFactory kf2 = KeyFactory.getInstance("RSA");
203             PrivateKey kf2.generatePrivate(spec2);
204             System.out.println("private key "+Base64.getEncoder().encodeToString(PrivateKey.getEncoded()));
205         }
206     }
207 }

```

```

288     catch(Exception ex){
289         System.err.println("No Private File"+ex.toString());
290     }
291 }
292 /**
293 * @param args the command line arguments
294 */
295 public static void main(String args[]) {
296
297     /* Set the Nimbus look and feel */
298
299     //editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) "
300     /* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and feel.
301      * For details see http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html
302      */
303     try {
304         for (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.getInstalledLookAndFeels()) {
305             if ("Nimbus".equals(info.getName())) {
306                 javax.swing.UIManager.setLookAndFeel(info.getClassName());
307                 break;
308             }
309         }
310     } catch (ClassNotFoundException ex) {
311         java.util.logging.Logger.getLogger(DecryptJFrame.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
312     } catch (InstantiationException ex) {
313         java.util.logging.Logger.getLogger(DecryptJFrame.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
314     } catch (IllegalAccessException ex) {
315         java.util.logging.Logger.getLogger(DecryptJFrame.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
316     } catch (java.awt.UnsupportedLookAndFeelException ex) {
317         java.util.logging.Logger.getLogger(DecryptJFrame.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
318     }
319 }
320 //
```

321
322
323 /* Create and display the form */
324 java.awt.EventQueue.invokeLater(new Runnable() {
325 public void run() {
326 new DecryptJFrame().setVisible(true);
327 }
328 });
329
330
331 try{
332 // to establish connection
333 socket = new Socket("localhost", 5000);
334 // object to read from socket
335 inputkey = new DataInputStream(socket.getInputStream());
336 inputtext = new DataInputStream(socket.getInputStream());
337 inptiv = new DataInputStream(socket.getInputStream());
338 // read key and use it in decryptButton method
339 keyinputkey.readUTF();
340 // read text and display it in cipher text field
341 CipherText= inputtext.readUTF();
342 // read IV as string then convert it as byte [] and use it in decryptButton method
343 iv= Base64.getDecoder().decode(inptiv.readUTF());
344 // display cipher text
345 CipherTextField.setText(CipherTextField.getText().trim()+"\n"+CipherText);
346
347
348 }
349 catch(Exception e){
350 System.err.println("no text receive"+e.toString());
351 }
352 }
353
354 // Variables declaration - do not modify
355 private javax.swing.JButton prKeyButton;
356 private javax.swing.JButton BackButton;
357 private static javax.swing.JButton DecryptButton;
358 private javax.swing.JLabel DecryptLabel;
359 private javax.swing.JLabel ErrorLabel;
360 static javax.swing.JTextField CipherTextField;
361 private javax.swing.JTextField PlainTextField;
362
363 // End of variables declaration
364 }
365

RSA class :

```
8 import java.security.*;
9 import java.util.Base64;
10 import javax.crypto.Cipher;
11 public class RSA {
12     // method to encrypt message by public key
13     public String encrypt ( String text , PublicKey pub) throws Exception {
14         byte [] TextByte =text.getBytes();
15         Cipher toCipher = Cipher.getInstance("RSA/ECB/PKCS1Padding");
16         toCipher.init(Cipher.ENCRYPT_MODE, pub);
17         byte [] cipherByte = toCipher.doFinal(TextByte);
18         return encode(cipherByte);
19     }
20     // method to decrypt cipher by private key
21     public String decrypt (String m , PrivateKey pr )throws Exception{
22         byte [] cipherByte = decode(m);
23         Cipher toPlain = Cipher.getInstance("RSA/ECB/PKCS1Padding");
24         toPlain.init(Cipher.DECRYPT_MODE,pr);
25         byte [] TextByte = toPlain.doFinal(cipherByte);
26         return new String(TextByte, "UTF8");
27     }
28     // method to convert byte[] to String
29     private String encode (byte [] s){
30         return Base64.getEncoder().encodeToString(s);
31     }
32     // method to convert String to Byte[]
33     private byte [] decode (String s){
34         return Base64.getDecoder().decode(s);
35     }
36 }
```

AES class :

```
8 import java.util.Base64;
9 import javax.crypto.*;
10 import javax.crypto.spec.GCMParameterSpec;
11 public class AES {
12     Cipher toCipher;
13     String IV;
14     //this method to encrypt the plain text for the sender.
15     public String encrypt (String Text, SecretKey key) throws Exception{
16         // convert text to byte[]
17         byte [] TextByte =Text.getBytes();
18         // to select encryption mode
19         toCipher = Cipher.getInstance("AES/GCM/NoPadding");
20         // Initialize cipher with secret key passed from parameter
21         toCipher.init(Cipher.ENCRYPT_MODE, key);
22         // store IV as String to share it through socket
23         IV=Base64.getEncoder().encodeToString(toCipher.getIV());
24         // start encryption
25         byte [] cipherByte = toCipher.doFinal(TextByte);
26         // convert cipher text to String
27         return encode(cipherByte);
28     }
29     //this method to decrypt the chipher text for the receiver.
30     public String decrypt (String cipherText ,SecretKey key, byte[] iv)throws Exception{
31         // convert text to byte[]
32         byte [] cipherByte = decode(cipherText);
33         // to select decryption mode
34         Cipher toPlain = Cipher.getInstance("AES/GCM/NoPadding");
35         GCMParameterSpec spec = new GCMParameterSpec(128,iv);
36         toPlain.init(Cipher.DECRYPT_MODE, key,spec);
37         byte [] TextByte = toPlain.doFinal(cipherByte);
38         return new String(TextByte);
39     }
40     // method to convert byte[] to String
41     private String encode (byte [] s){
42         return Base64.getEncoder().encodeToString(s);
43     }
44     // method to convert String to Byte[]
45     private byte [] decode (String s){
46         return Base64.getDecoder().decode(s);
47     }
48 }
49 ]
```

userInfo class :

```
7 import java.io.*;
8 import java.nio.charset.StandardCharsets;
9 import java.security.NoSuchAlgorithmException;
10 import java.security.*;
11 import java.security.spec.PKCS8EncodedKeySpec;
12 import java.security.spec.X509EncodedKeySpec;
13 import java.util.Base64;
14
15 public class userInfo {
16     String userName ;
17     String password ;
18     private PublicKey pubKey ;
19     private PrivateKey prKey;
20     // to create instance from the class
21     public userInfo(String userName, String password) throws Exception {
22         this.userName = userName;
23         // send password to save hashed password
24         this.password = generateHash(password);
25         // generate private and public keys
26         KeyPairGenerator kgenerator = KeyPairGenerator.getInstance("RSA");
27         kgenerator.initialize(1024);
28         KeyPair pair=kgenerator.generateKeyPair();
29         prKey=pair.getPrivate();
30         pubKey=pair.getPublic();
31         // Store Public Key.
32         X509EncodedKeySpec x509EncodedKeySpec = new X509EncodedKeySpec(pubKey.getEncoded());
33         FileOutputStream keysFile = new FileOutputStream(this.userName + "public.key");
34         keysFile.write(x509EncodedKeySpec.getEncoded());
35         keysFile.close();
36         // Store Private Key.
37         PKCS8EncodedKeySpec pkcs8EncodedKeySpec = new PKCS8EncodedKeySpec(prKey.getEncoded());
38         keysFile = new FileOutputStream(this.userName + "private.key");
39         keysFile.write(pkcs8EncodedKeySpec.getEncoded());
40         keysFile.close();
41     }
42     public String getUserName() {
43         return userName;
44     }
45     public String getPassword() {
46         return password;
47     }
48     public PublicKey getPubKey() {
49         return pubKey;
50     }
51     public PrivateKey getPrKey() {
52         return prKey;
53     }
54     // method to hash the password using SHA-512
55     public String generateHash(String password_text) throws Exception {
56         MessageDigest md = MessageDigest.getInstance ("SHA-512");
57         byte[] hashedPassword = md.digest(password_text.getBytes(StandardCharsets.UTF_8));
58         // convert hashed password to String
59         return Base64.getEncoder().encodeToString(hashedPassword);
60     }
61 }
```

4.0 Challenges:

1. One of the challenges that we faced, we did not know how to make the user send a file to the other user ‘receiver’, so we found an alternative solution Make the user send a text message.
2. We could not be able to read the cipher ,symmetric key and IV separately. So, we create input stream objects for each one.
3. We had also a problem with method decryption in the AES class that didn't work .To solve this problem, we saved the IV in a variable and sent it to the receiver.
4. To send the key through the socket it has to be converted to string. So, we face a problem to return it back as a SecretKey. Also reading private and public key from a file will return a String , so we had to convert the to PublicKey or PrivateKey.

5.0 References:

1. <https://www.youtube.com/watch?v=kqBmsLvWU14>
2. <https://www.youtube.com/watch?v=8wlE6DgOWBs>
3. <https://youtu.be/R9eerqP78PE>
4. <https://youtu.be/J1RmZZEkNOk>
5. <https://stackoverflow.com/questions/5355466/convert-secret-key-into-a-string-and-vice-versa>
6. <https://snipplr.com/view/18368/saveload--private-and-public-key-tofrom-a-file>
7. <https://www.snippets.com/java/util/java-convert-string-to-rsa-public-key/>
8. <https://www.geeksforgeeks.org/sha-512-hash-in-java/>