REVERSIBLE DATA HIDING USING ENCRYPTED BYTE ARRAY IMAGE

Authors Name/s: A. Vigneysh Aravindh
Department of Computer Science and Engineering
S.R.M University
Chennai – 600026, India
Email ID: vigneysharavindh@gmail.com

ABSTRACT

Outsourced storage of Multimedia Files by cloud has become more and more popular. To Manage the Outsourced Image, the cloud server may embed some additional information regarding the ownership of Image. Obviously, the cloud service has no rights to introduce permanent Data Distortion. Different from all previous encryption-based frameworks, in which the cipher texts may attract the notation of the curious cloud, RIT-based framework allows the user to transform the content of original image into the content of another target image with the same size. Therefore, RDH is needed. It is a technique by which the original image can be recovered without loss after the embedded message is extracted. Cloud service makes it challenging to protect privacy of the image contents. Encryption is the most popular technique for protecting privacy. The cloud server can easily embed data into the 'encrypted image' by any RDH methods for plaintext images and thus a client-free scheme for RDH-EI can be realized , that is, the data-embedding process executed by the cloud server is irrelevant with processes of both encryption and decryption. Hence, RDH is implemented in encrypted image so that the cloud server can reversibly embed data but cannot receive any information about the original image.

1. INTRODUCTION

1.1 GENERAL

Steganography is the practice of concealing a file, message, image, or video within another file, message, image, or video. The advantage of steganography over cryptography alone is that the intended secret message does not attract attention to itself as an object of scrutiny. Plainly visible encrypted messages—no matter how unbreakable—arouse interest, and may in themselves be incriminating in countries where encryption is illegal. Thus, whereas cryptography is the practice of protecting the contents of a message alone, steganography is concerned with concealing the fact that a secret message is being sent, as well as concealing the contents of the message.

1.2 OBJECTIVE

The Main Objective is to encrypt an Image Before performing the Reversible Image Transformation Technique. Adding a Second Layer of Security Enhances the Privacy offered to the Images that are going to be outsourced to a Server. Using this system data extraction and image recovery are free of any error. DE technique is still a promising technique and it is expected that more serious trials to improve it will be revealed during the next few years.

1.3 DESCRIPTION

Reversible data hiding is a type of data hiding techniques whereby the host image can be recovered exactly. Being lossless makes this technique suitable for medical and military applications. Difference expansion (DE) is one of the most important techniques which are used for reversible data hiding. This technique received more attention over the years because of its high efficiency and simplicity.

The aim of this paper is to present a review of reversible DE-based data hiding techniques proposed so far in order to define the purpose of DE-based data hiding, reflecting recent progress, and provide some research issues for the future. Many researchers tried to improve its performance in terms of hiding capacity and visual perceptibility. However, some of the modified techniques need more improvements to be applicable. Steganography includes the concealment of information within computer files. In digital steganography, electronic communications may include steganographic coding inside of a transport layer, such as a document file, image file, program or protocol.

Media files are ideal for steganographic transmission because of their large size. For example, a sender might start with an innocuous image file and adjust the color of every 100th pixel to correspond to a letter in the alphabet, a change so subtle that someone not specifically looking for it is unlikely to notice it Reversible data hiding scheme is the technique that allows embedding data inside an image and later the hidden data can be retrieved as required and the exact copy of the original host image is recovered. Some of the traditional reversible data hiding schemes are based on modulo-arithmetic additive and spread-spectrum techniques.

An example of these schemes is the one proposed by Honsinger , which based on addition of modulo 256 as an invertible operation. Although some of these schemes are robust, the modulo-arithmetic-based reversible data-hiding algorithms have the disadvantage of salt-and-pepper visual artifacts and hinder watermark retrieval. In order to enhance the robustness of the reversible watermarking and reduce the salt-and-pepper visual artifact of the above mentioned schemes, histogram shifting techniques were proposed. In this scheme, the embedding target is replaced by the histogram of a block of the image. A good example of the scheme is the circular interpretation scheme proposed by Vleeschouwer. Although this type of data hiding schemes provides a higher quality of the embedded image, the embedding capacity is lower. A different category of data hiding schemes involves methods that losslessly compress a set of selected features from an image and embed the payload in the space saved due to the compression. This type results in higher embedding capacity than the previously mentioned types. Another scheme of this type is the generalized Least Significant Bit (g-LSB) embedding algorithm proposed by Celik , which is based on grouping the pixels and embedding data bits into the state of each group.

2. SYSTEM ANALYSIS

2.1 LITERATURE SURVEY

Title: - Reversible Data Hiding using PSNR and RDH technique

Author: Asha S Raj, Mrs. Gayathri Nair P

Year: 2016

Description: A reversible data hiding can recover the original image without any distortion from the marked image after the hidden data have been extracted. The techniques utilizes the zero or the minimum points of the histogram of an image. It modifies the pixel values to embed data into the image. It can embed more data than many of the existing reversible data hiding techniques. It is proved that the peak signal-to-noise ratio (PSNR) of the marked image generated by the method. The PSNR is higher for all reversible data hiding techniques. Reversible Data Hiding is used to embed a piece of information into the host images to generate the marked one. Original image can be exactly recovered after extracting the embedded data. The original cover can be reversibly restored after the embedded information is extracted. The Reversible Data Hiding process eliminates the disadvantages of reversible watermarking. The process to reverse the marked images back to the original cover images after the hidden data are extracted. The process which can be used the Peak Signal Noise Ratio (PSNR) to check the quality of reversed image. The PSNR value is the maximum possible power of a signal and the power of corrupting noise. It is the most commonly used measure of quality of reconstruction. PSNR represent

the distortion level between marked image and cover image. Reversible Data Hiding mostly used Difference expansion (DE).It is one of the most important techniques which are used for reversible data hiding. With the help of PSNR value the recovered images can be checked.

Title: - Image distortion and Target compression
Author: K-.S.Kim, M.-J.Lee,H.-Y.Lee
Year: 2014
Description: reversible data hiding scheme for perfectly restored the original content after extraction of the hidden data. That is the optimal rule of value modification under a payload-distortion criterion is found by using an iterative procedure. The secret data, as well as the auxiliary information are carried by the differences between the original pixel-values and the corresponding values estimated from the neighbors. Here, the optimal value transfer rule is used to modify the estimation errors. The optimal value transfer matrix is produced for maximizing the amount of secret data, i.e., the pure payload, by the iterative procedure. Also, the host image is divided into a number of pixel subsets and an estimation error in the next subset is always embedded by the auxiliary information of a previous subset. A receiver can successfully extract the embedded secret data and recover the original content in the subsets with an reverse order. In this way, a good reversible practical data hiding performance is achieved. This way, a good payload-distortion performance can be achieved.

Title: - A new Reversible Data Hiding Scheme with Improved Capacity Based on Directional Interpolation and Difference Expansion
Author: P.V Sabeen Govind, M. Wilscy
Year: 2015
Description: Using reversible data hiding (RDH) we can hide our secret data into a cover image and the receiver can restore both the secret data and the original image. It has wide application in medical imagery, military imagery where no distortion of original cover is allowed. Hong and Chen proposed a RDH scheme based on interpolation and histogram shifting. In their scheme reference pixels are not used for data embedding which leads to low capacity. Huang *et al.* modified this scheme and proposed a high capacity RDH scheme in which prediction errors are used for data embedding. In this paper we propose a further modification to the scheme of Huang *et al.* based on directional interpolation. Directional interpolation yields a better approximation to the original pixel which improves the capacity of embedding.

2.2 EXISTING SYSTEM

In this framework, a content owner encrypts the original image using a standard cipher with an encryption key. After producing the encrypted image, the content owner hands over it to a data hider (e.g., a database manager) and the data hider can embed some auxiliary data into the encrypted image by losslessly vacating some room according to a data hiding key. Then a receiver, maybe the content owner himself or an authorized third party can extract the embedded data with the data hiding key and further recover the original image from the encrypted version according to the encryption key

2.2.1 EXISTING TECHNIQUE

- Reversible data hiding (RDH)

DISADVANTAGES

- All previous methods embed data by reversibly vacating room from the encrypted images, which may be subject to some errors on data extraction and/or image restoration.
- It is difficult for data hider to reversibly hide the data behind the image.

2.3 PROPOSED SYSTEM

Since losslessly vacating room from the encrypted images is relatively difficult and sometimes inefficient, why are we still so obsessed to find novel RDH techniques working directly for encrypted images? If we reverse the order of encryption and vacating room, i.e., reserving room prior to image encryption at content owner side, the RDH tasks in encrypted images would be more natural and much easier which leads us to the novel framework, "reserving room before encryption (RRBE)".Obviously, standard RDH algorithms are the ideal operator for reserving room before encryption and can be easily applied to Framework RRBE to achieve better performance compared with techniques from Framework VRAE.

2.2.1 PROPOSED TECHNIQUE

- Reversible Data Hiding using Encrypted Byte Array Image

ADVANTAGES

- In this system it uses traditional RDH algorithm, and thus it is easy for the data hider to reversibly embed data in the encrypted image.
- Using this system data extraction and image recovery are free of any error.

3. PROJECT DESCRIPTION

3.1 GENERAL

We get the Image from the user and encrypt it for Security purposes and embed the Data into a Target Image to Maintain Confidentiality.

3.2 PROBLEM DEFINITION

The Online File Storage Services are Vulnerable to Many Security Threats. The Services, Once hacked, gives access to the Attacker of all our Private Content. Recent Attack on Apple's Cloud Storage Service known as iCloud is a Prime Example. Many Celebrity's Private Pictures have been leaked, leading to a Massive Controversy. These Online Services, when processing our Images, are specifically curious when they come across Encrypted Images. They're designed to take Interest in these types of Images, there by attracting Attention of the Hacker who gains Access. The fact that Services know that your Image is encrypted, itself in a way is a Violation of Privacy.

3.3 METHODOLOGIES

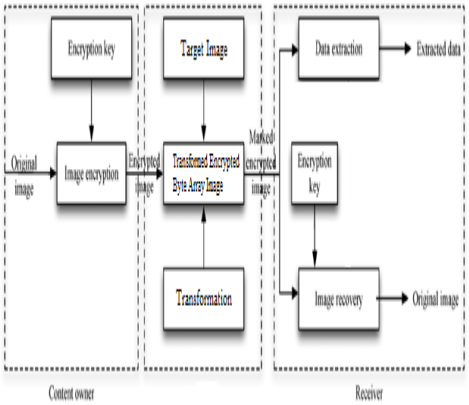
The Image to be Encrypted and Transformed is encrypted with the Use of DES Algorithm and a 64 Bit Symmetric Key given by the User. The Encrypted Image is converted to byte Array Format which is embedded into the Target Image. To retrieve the Original Image, The Picture is Reverse Transformed and Decrypted.

3.3.1 MODULE & MODULE CONTENT

MODULES

- **Authentication Module**
  - **Sign In Module**
  - **Sign Up Module**
- **Cryptography Module**
  - **Encryption Module**
  - **Decryption Module**
- **Transformation Module**
  - **Image Transform Module**

- **Image Reverse Transform Module**



4. SYSTEM REQUIREMENTS

4.1 HARDWARE REQUIREMENTS

Processor : Intel Core i3-5005u

Hard Disk : 512 GB HDD

RAM : 3 GB

4.2 SOFTWARE REQUIREMENTS

Operating System : Windows 8.1

IDE : NetBeans 8.2

Front End : Java

SYSTEM DESIGN

5.1 GENERAL

System design is a process to transform user requirements into somesuitable form, which helps the programmer in software coding and implementation.

For assessing user requirements, an SRS (Software Requirement Specification) document is created whereas for coding and implementation, there is a need of more specific and detailed requirements in software terms. The output of this process can directly be used into implementation in programming languages.

Software design is the first step in SDLC (Software Design Life Cycle), which moves the concentration from problem domain to solution domain. It tries to specify how to fulfill the requirements mentioned in SRS.

5. SYSTEM DESIGN

Fig.1.0 System Architecture

Fig.1.1 Deployment Diagram

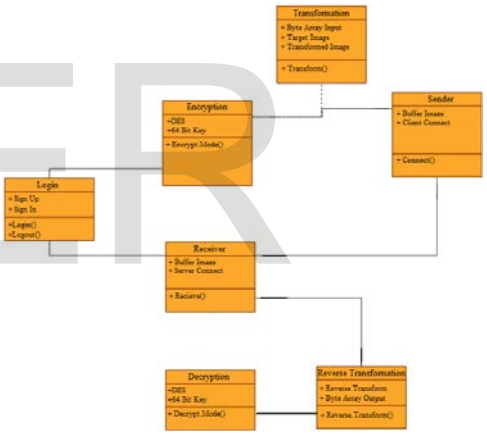


Fig.1.2 Use Case Diagram

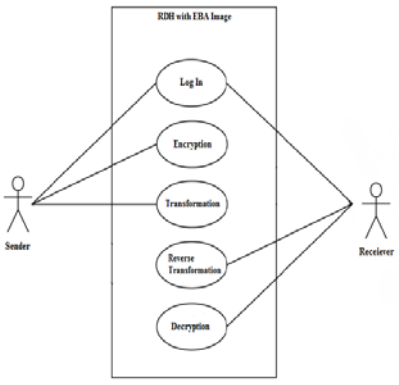


Fig.1.3 Class Diagram

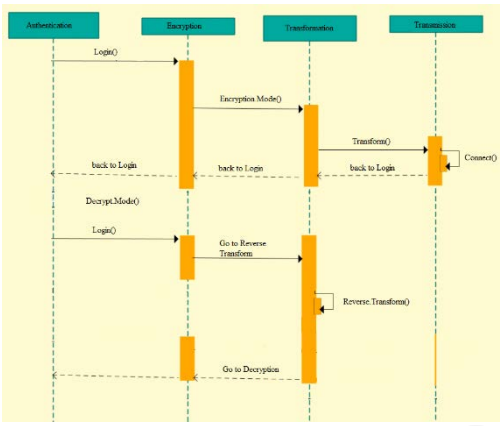


Fig1.5 Sequence Diagram

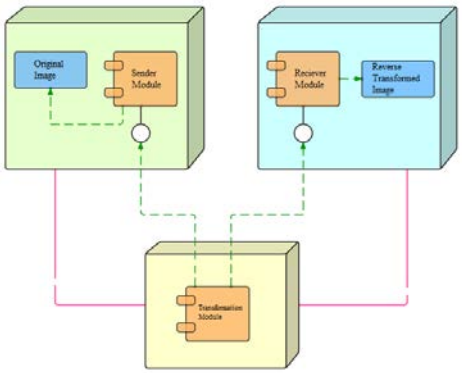
6. SOFTWARE TESTING

6.1 GENERAL

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub-assemblies, assemblies and/or a finished product. It is the process of exercising software with the intent of ensuring that the Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of tests. Each test type addresses a specific testing requirement.

6.2 DEVELOPING METHODOLOGIES

The test process is initiated by developing a comprehensive plan to test the general functionality and special features on a variety of platform combinations. Strict quality control procedures are



used.

The process verifies that the application meets the requirements specified in the system requirements document and is bug free. The following are the considerations used to develop the framework from developing the testing methodologies.

6.3 TYPES OF TESTS

6.3.1 Unit Test

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program input produces valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application. It is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

6.3.2 Functional Test

Functional tests provide systematic demonstrations that functions tested are available as specified by the

business and technical requirements, system documentation, and user manuals.

Functional testing is centered on the following items:

Valid Input : identified classes of valid input must be accepted

Invalid Input : identified classes of invalid input must be rejected

Functions : identified functions must be exercised

Output : identified classes of application outputs must be exercised

Systems/Procedures : interfacing systems or procedures must be invoked.

6.3.3 System Test

System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration oriented system integration test. System testing is based on process descriptions and flows, emphasizing pre-driven process links and integration points.

6.3.4 Performance Test

The performance test ensures that the output be produced within the time limits, and the time taken by the system for compiling, giving response to the users and request being send to the system for to retrieve the results.

6.3.5 Integration Test

Software integration testing is the incremental integration testing of two or more integrated software components on a single platform to produce failures caused by interface defects.

6.3.6 Acceptance Test

User Acceptance Testing is a critical phase of any project and requires significant participation by the end user. It also ensures that the system meets the functional requirements.

Acceptance Testing for Data Synchronization:

- The Acknowledgements will be received by the Sender Node after the Packets are received by the Destination Node.
- The Route add operation is done only when there is a Route request in need
- The status of Nodes information is done automatically in the Cache Updating process.

6.3.7 Build the test plan

Any project can be divided into units that can be further performed for detailed processing. Then a testing strategy for each of this unit is carried out. Unit testing helps to identify the possible bugs in the individual component, so the component that has bugs can be identified and can be rectified from errors.

7. APPLICATION

7.1 GENERAL

This session gives the details of our application usage in secure Image transmission and it's Usefulness.

7.2. APPLICATION

USE IN MODERN PRINTERS

The larger the Target Image (in binary data, the number of bits) relative to the hidden Encrypted Image, the easier it is to hide the latter. For this reason, digital pictures (which contain large amounts of data) are used to hide messages on the Internet and on other communication media. It is not clear how common this actually is. For example: a 24-bit bitmap uses 8 bits to represent each of the three colour values (red, green, and blue) at each pixel. The blue alone has 28 different levels of blue intensity. The difference between 11111111 and 11111110 in the value for blue intensity is likely to be undetectable by the human eye. Therefore, the least significant bit can be used more or less undetectably for something else other than colour information. If this is repeated for the green and the red elements of each pixel as well, it is possible to encode one byte of Encrypted Image for every three pixels. Some modern computer printers use steganography,

including HP and Xerox brand colour laser printers. These printers could print the Dots which are Actually the Encrypted Byte Array which can be recovered on scanning the Image.

USE IN MOBILE APPLICATIONS

Can be used as an Android / iOS Application to to secure images so that if Users do not want third party apps or users to access their content, they could make use of the App to Encrypt and Transform the Image into something that is irrelevant to the context of the Original Image. That way, the third party Apps or users will not be able to guess that the image has been transformed,hence they wouldn't be interested in that Image at all. Lots of Online Blackmailing and abuse can be prevented with the safety features this application provides. It's a viable method to ensure Privacy and Protection of personal Content.

8. CONCLUSION

8.1 FUTURE ENHANCEMENT

As a part of our Future Work, This Project holds Great Potential for Privacy Issues and Copyright Issue Prevention for not only Images but other Multimedia Files such as Audio and Video. As on Example of a Future Enhancement, We Planned to Hide and Audio File in a Target Audio file irrelevant to the Original File. Enveloping Technique, Same technique used to Transmit Radio Signals can be used to Embed the Message Signal in another Message Signal with gets enveloped to the Carrier Signal.

8.2 SUMMARY

We introduced an Image Steganography Method which Ensures Privacy and Prevention of Online Abuse and Blackmail. The Algorithm Primarily Focuses on Cryptography for Security and Reversible Data Hiding for Confidentiality and Integrity of the Data. It is a Simple Application with lots of Room for Enhancements and Potential for serving a better Purpose for the Online Society.

REFERENCES

[1] K. Hwang, D. Li, "Trusted cloud computing with secure resources and data coloring," IEEE Internet Computing, vol. 14, no. 5, pp. 14-22, Sept.-Oct. 2010.

[2] F. Bao, R. H. Deng, B. C. Ooi, et al., "Tailored reversible watermarking schemes for authentication of electronic clinical atlas," IEEE Trans. on Information Technology in Biomedicine, vol. 9, no. 4, pp. 554-563, Dec. 2005.

[3] F. Willems, D. Maas, and T. Kalker, "Semantic lossless source coding," 42nd Annual Allerton Conference on Communication, Control and Computing, Monticello, Illinois, USA, pp. 1411-1418, 2004.

[4] W. Zhang, X. Hu, N. Yu, et al. "Recursive histogram modification: establishing equivalency between reversible data hiding and lossless data compression," IEEE Trans. on Image Processing, vol. 22, no. 7, pp. 2775-2785, Jul. 2013.

[5] V. Sachnev, H. J. Kim, J. Nam, S. Suresh, and Y. Q. Shi, "Reversible watermarking algorithm using sorting and prediction," IEEE Trans. on Circuits and Systems for Video Technology, vol.19, no.7, pp. 989-999, Jul. 2009.

[6] B.ou, X. Li, Y. Zhao, R. Ni, Y. Shi, "Pairwise prediction-error expansion for efficient reversible data hiding," IEEE Trans. on Image Processing, vol. 22, no.12, pp. 5010-5021, Dec. 2013.

[7] Ioan-Catalin Dragoi, Dinu Coltuc, "Local-prediction-based difference expansion reversible watermarking," IEEE Trans. on Image Processing, vol. 23, no. 4, pp. 1779-1790, Apr. 2014.

[8] Z. Ni, Y. Shi, N. Ansari, and S. Wei, "Reversible data hiding," IEEE Trans. on Circuits and Systems for Video Technology, vol. 16, no. 3, pp. 354-362, Mar. 2006.

[9] J. Tian, "Reversible data embedding using a difference expansion," IEEE Trans. on Circuits and Systems for Video Technology, vol. 13, no.8, pp. 890-896, Aug. 2003.

[10] X. Hu, W. Zhang, X. Li, N. Yu, “Minimum rate prediction and optimized histograms modification for reversible data hiding,” IEEE Trans. on Information Forensics and Security, vol. 10, no. 3, 653-664, Mar. 2015.

APPENDIX 1

SAMPLE CODING

Encryption

```
private void
jButton3ActionPerformed(java.awt.event.ActionEvent
evt) {
   try{
      FileInputStream file = new
FileInputStream(T1.getText());
      FileOutputStream outStream = new
FileOutputStream("Encrypted Image.png");
      String KeyString = TB1.getText();
      byte k[];
      k = KeyString.getBytes();
      SecretKeySpec key = new SecretKeySpec(k,
"DES");
      Cipher enc = Cipher.getInstance("DES");
      enc.init(Cipher.ENCRYPT_MODE, key);
      CipherOutputStream cos = new
CipherOutputStream(outStream,enc);
      byte[] buf = new byte[196608];
      int read;
      while((read=file.read(buf))!=-1)
      {
         cos.write(buf,0,read);
      }
      int width = 128;
      int height = 128;

      DataBuffer buffer = new DataBufferByte(buf,
buf.length);

      //3 bytes per pixel: red, green, blue
      WritableRaster raster =
Raster.createInterleavedRaster(buffer, width, height, 3
* width, 3, new int[] {0, 1, 2}, (Point)null);
      ColorModel cm = new
ComponentColorModel(ColorModel.getRGBdefault().
getColorSpace(), false, true, Transparency.OPAQUE,
DataBuffer.TYPE_BYTE);
      BufferedImage image = new BufferedImage(cm,
raster, true, null);
      ImageIO.write(image, "png", new File("Converted
Noise Image.jpg"));

      file.close();
      outStream.flush();
      cos.close();
      jOptionPane1.showMessageDialog(null, "The
Image Was Encrypted Succesfully!");
   }catch(Exception e){
      jOptionPane1.showMessageDialog(null,e);
   }
   }
```

Decryption

```
try{
      FileInputStream file = new
FileInputStream(T1.getText());
      FileOutputStream outStream = new
FileOutputStream("Decrypted Image.png");
      String KeyString = TB1.getText();
      byte k[];
      k = KeyString.getBytes();
      SecretKeySpec key = new SecretKeySpec(k,
"DES");
      Cipher enc = Cipher.getInstance("DES");
      enc.init(Cipher.DECRYPT_MODE, key);
      CipherOutputStream cos = new
CipherOutputStream(outStream,enc);
      byte[] buf = new byte[1024];
      int read;
      while((read=file.read(buf))!=-1)
      {
         cos.write(buf,0,read);
      }
      file.close();
      outStream.flush();
      cos.close();
      jOptionPane1.showMessageDialog(null, "The
Image Was Decrypted Succesfully!");
   }catch(Exception e){
      jOptionPane1.showMessageDialog(null,e);
   }
```

Transformation

```
private void
jButton2ActionPerformed(java.awt.event.ActionEvent
evt) {
   try{
      FileInputStream file = new
FileInputStream(T1.getText());
      FileOutputStream outStream = new
FileOutputStream("Decrypted Image.png");
      String KeyString = TB1.getText();
      byte k[];
      k = KeyString.getBytes();
      SecretKeySpec key = new SecretKeySpec(k,
"DES");
      Cipher enc = Cipher.getInstance("DES");
      enc.init(Cipher.DECRYPT_MODE, key);
      CipherOutputStream cos = new
CipherOutputStream(outStream,enc);
      byte[] buf = new byte[1024];
      int read;
      while((read=file.read(buf))!=-1)
      {
         cos.write(buf,0,read);
      }
      file.close();
      outStream.flush();
      cos.close();
      jOptionPane1.showMessageDialog(null, "The
Image Was Decrypted Succesfully!");
   }catch(Exception e){
      jOptionPane1.showMessageDialog(null,e);
```

```
    }
    }
```

Reverse Transformation

```
private void
jButton3ActionPerformed(java.awt.event.ActionEvent
evt) {
     File Input = new File(T1.getText());
     BufferedImage Image;
      try {
         byte[] op = null;
         String s = null;
         Image = ImageIO.read(Input);
         TA2.append("Decoding");
         int w=Image.getWidth(),h=Image.getHeight();
         int msglength=(Image.getRGB(0, 0)&0xff);
         TA2.append("\nByte Buffer Length:
"+msglength);
         for(int row=0,j=0,i=1;   row<h   ;row++  )
         {
          for(int col=0;col<w && j<msglength ;col++
,i++)
          {
           if (i%11==0) {
                     int result=Image.getRGB(col,row);
                     int charatpos = (((result >> 16) &
0x7) << 5);
                     charatpos |=  (((result >> 8) & 0x7)
<< 2);
                     charatpos |=  ((result & 0x3));
                     s = s+(char)charatpos;
                     j++;
                    }
          }
         }
         op = s.getBytes();
         File file = new File(T4.getText());
         File fos = new File("Reverse
Transformed.png");
         FileInputStream is = null;
         FileOutputStream os = null;
         try {
             is = new FileInputStream(file);
             os = new FileOutputStream(fos);
             byte[] buffer = new byte[1024];
             int length;
             while ((length = is.read(buffer)) > 0) {
             os.write(buffer, 0, length);
      }
    } finally {
      is.close();
      os.close();
    }
         TA2.append("\nDecoding done!");
         }
         catch (IOException ex) {
         TA2.append("\n"+ex.getMessage());
                          }

    }
```

```
    private void
jButton4ActionPerformed(java.awt.event.ActionEvent
evt) {
        jFileChooser1.showOpenDialog(null);
        File f=jFileChooser1.getSelectedFile();
        String filename1=f.getAbsolutePath();
        T4.setText(filename1);
    }
```

Server – Sending Image

```
private void
jButton1ActionPerformed(java.awt.event.ActionEvent
evt) {
          ServerSocket server=null;
          Socket socket = null;
        try {
           server = new ServerSocket(4000);
        } catch (IOException ex) {

Logger.getLogger(Server_Image.class.getName()).log(
Level.SEVERE, null, ex);
        }
          TA1.append("Server Waiting for image");

        try {
           socket = server.accept();
        } catch (IOException ex) {

Logger.getLogger(Server_Image.class.getName()).log(
Level.SEVERE, null, ex);
        }
          TA1.append("\nClient connected.");

          InputStream in = null;
        try {
           in = socket.getInputStream();
        } catch (IOException ex) {

Logger.getLogger(Server_Image.class.getName()).log(
Level.SEVERE, null, ex);
        }
          DataInputStream dis = new
DataInputStream(in);

          int len = 0;
        try {
           len = dis.readInt();
        } catch (IOException ex) {

Logger.getLogger(Server_Image.class.getName()).log(
Level.SEVERE, null, ex);
        }
          TA1.append("\nImage Size: " + len/1024 +
"KB");

          byte[] data = new byte[len];
        try {
           dis.readFully(data);
        } catch (IOException ex) {

Logger.getLogger(Server_Image.class.getName()).log(
Level.SEVERE, null, ex);
```

```
        }
        try {
           dis.close();
        } catch (IOException ex) {

Logger.getLogger(Server_Image.class.getName()).log(
Level.SEVERE, null, ex);
        }
        try {
           in.close();
        } catch (IOException ex) {

Logger.getLogger(Server_Image.class.getName()).log(
Level.SEVERE, null, ex);
        }

          InputStream ian = new
ByteArrayInputStream(data);
          BufferedImage bImage = null;
        try {
           bImage = ImageIO.read(ian);
        } catch (IOException ex) {

Logger.getLogger(Server_Image.class.getName()).log(
Level.SEVERE, null, ex);
        }

          ImageIcon icon=new ImageIcon(bImage);
          L1.setIcon(icon);       // TODO add your
handling code here:
    }
```

Client – Receiving Image

```
private void
jButton2ActionPerformed(java.awt.event.ActionEvent
evt) {
    Socket soc = null;
      BufferedImage img = null;
      try {
        soc=new Socket("localhost",4000);
       }
      catch (IOException ex) {

Logger.getLogger(Client_Image.class.getName()).log(
Level.SEVERE, null, ex);
       }
      System.out.println("Client is running. ");

     try {

        TA1.append("Reading image from disk. ");
        img = ImageIO.read(new File(T1.getText()));
        ByteArrayOutputStream baos = new
ByteArrayOutputStream();

        ImageIO.write(img, "jpg", baos);
        baos.flush();

        byte[] bytes = baos.toByteArray();
        baos.close();

        TA1.append("\nSending image to server. ");

        OutputStream out = soc.getOutputStream();
        DataOutputStream dos = new
DataOutputStream(out);

        dos.writeInt(bytes.length);
        dos.write(bytes, 0, bytes.length);

        TA1.append("\nImage sent to server. ");

        dos.close();
        out.close();

      }catch (Exception e) {
        TA1.append("\nException: " + e.getMessage());
          try {
             soc.close();
          } catch (IOException ex) {

Logger.getLogger(Client_Image.class.getName()).log(
Level.SEVERE, null, ex);
          }
      }
         try {
            soc.close();
         } catch (IOException ex) {

Logger.getLogger(Client_Image.class.getName()).log(
Level.SEVERE, null, ex);
          }
     }
```

Login

```
private void
jButton1ActionPerformed(java.awt.event.ActionEvent
evt) {
String Username;
String Pass; // Declaring Variables

Username = T1.getText();
Pass = P1.getText();

if(Username.equals("admin") &&
Pass.equals("admin"))
{
   Op1.showMessageDialog(null, "Login Succesful!");
   Image_Chooser IG = new Image_Chooser();
   IG.setVisible(true);
   this.dispose();
}
else if(Username.equals("rec")&&Pass.equals("rec"))
{
   Op1.showMessageDialog(null, "Login Succesful!");
   Server_Image IG = new Server_Image();
   IG.setVisible(true);
   this.dispose();
}
else
{
   Op1.showMessageDialog(null, "Login Failed!");
}
   }
```

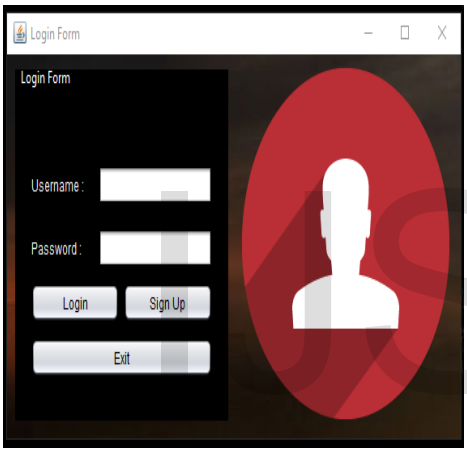
Choosing and Displaying an Image

```
private void
jButton1ActionPerformed(java.awt.event.ActionEvent
evt) {
    jFileChooser1.showOpenDialog(null);
    File f=jFileChooser1.getSelectedFile();
    String filename=f.getAbsolutePath();
    T1.setText(filename);
    ImageIcon icon=new ImageIcon(filename);
    L1.setIcon(icon);
    }
```

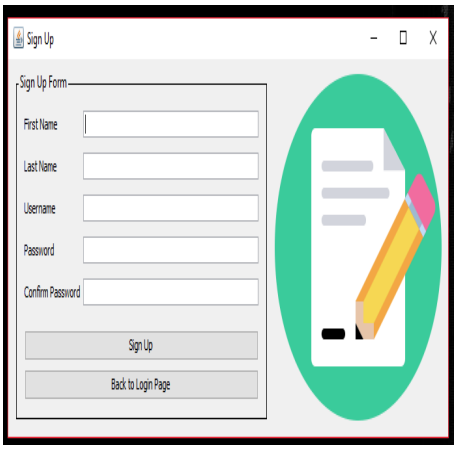
APPENDIX 2

SCREENSHOTS

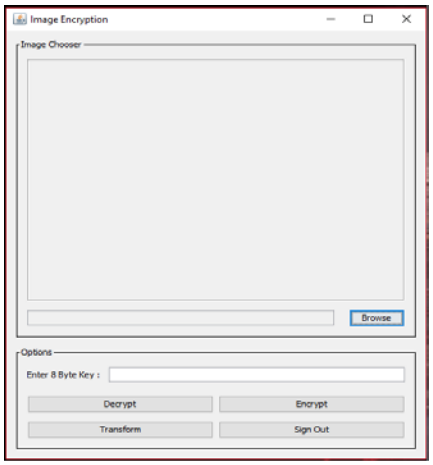
Login



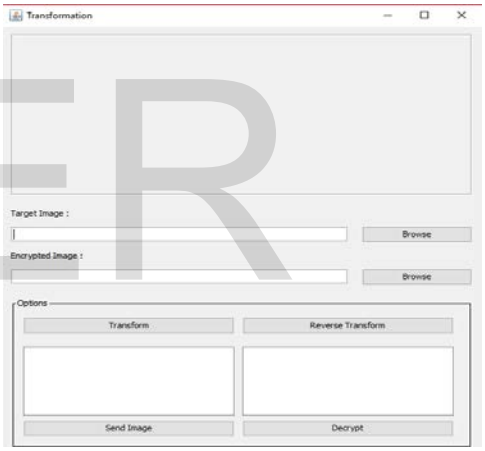
Sign Up



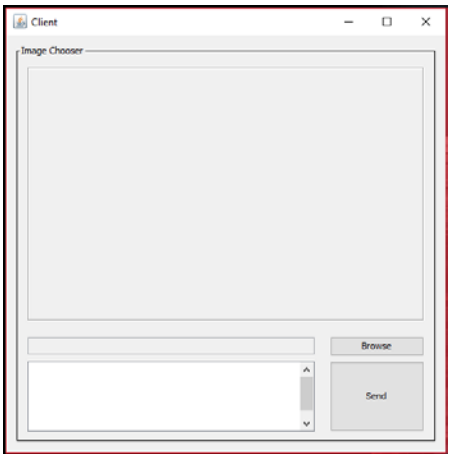
Encryption / Decryption Module

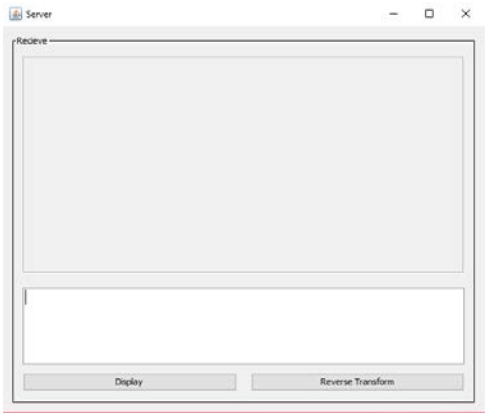


Transformation / Reverse Transformation Module



Sender Module





Receiver Module