Implementation of Secure Payment Transaction using AES encryption with extended Visual Cryptography

GRADUATE PROJECT REPORT

Submitted to the Faculty of
the Department of Computing Sciences
Texas A&M University-Corpus Christi
Corpus Christi, Texas

In Partial Fulfillment of the Requirements for the Degree of
Master of Science in Computer Science

By

Naveen Kumar Kolli
Summer 2017

Committee Members

Dr. Mario Garcia
Committee Chairperson

Dr. Ajay Katangur
Committee Member
ABSTRACT

Web application security has become an important concern for every user. This project implements a secure transaction platform that helps users to make secure bank transactions. In the communication between bank and merchant, every time merchant must be verified to prevent fraudulent transactions.

In this project, when a user requests a transaction from a bank, the bank generated one-time pass code (OTP) will be converted into a quick response (QR) code, a matrix barcode that allows quick access to URLs, among other uses. The QR code image is converted into two different shares using (2, 2) scheme of visual cryptography. (2, 2) scheme of visual cryptography is a technique in which two image shares are created out of one image. Image shares are components of the original image so that any one share (on its own) does not reveal anything about the original image. Only if you have all of the shares can you determine the secret image. One share is given to the merchant server with an advanced encryption standard (AES) encrypted image. Only the legitimate merchant server can access the share because the AES key is required to decrypt the share. Another share will be sent to the users registered email address. Both shares are overlaid to obtain the original QR code. QR code is scanned by the user to obtain the one time pass code. OTP is entered in the merchant server web page to complete the transaction. This project is an implementation of online fraud prevention using secure authentication process, visual cryptography, and encryption.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>CHAPTER</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABSTRACT</td>
<td>ii</td>
</tr>
<tr>
<td>TABLE OF CONTENTS</td>
<td>iii</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>v</td>
</tr>
<tr>
<td>1 BACKGROUND AND RATIONALE</td>
<td>1</td>
</tr>
<tr>
<td>1.1 Image Steganography</td>
<td>2</td>
</tr>
<tr>
<td>1.1.1 LSB embedding</td>
<td>2</td>
</tr>
<tr>
<td>1.2 Encryption and Decryption Techniques</td>
<td>4</td>
</tr>
<tr>
<td>1.2.1 Advanced Encryption Standard</td>
<td>4</td>
</tr>
<tr>
<td>1.3 Phishing Process</td>
<td>5</td>
</tr>
<tr>
<td>1.4 Visual Cryptography and QR Code</td>
<td>5</td>
</tr>
<tr>
<td>1.5 Existing Techniques</td>
<td>6</td>
</tr>
<tr>
<td>1.6 Motivation</td>
<td>9</td>
</tr>
<tr>
<td>1.7 Project Objective</td>
<td>10</td>
</tr>
<tr>
<td>2 SYSTEM DESIGN AND ARCHITECTURE</td>
<td>11</td>
</tr>
<tr>
<td>3 SYSTEM IMPLEMENTATION</td>
<td>13</td>
</tr>
<tr>
<td>3.1 System Requirements</td>
<td>13</td>
</tr>
<tr>
<td>3.1.1 Software Requirements</td>
<td>13</td>
</tr>
<tr>
<td>3.1.2 Hardware Requirements</td>
<td>13</td>
</tr>
<tr>
<td>3.2 Banking Application module</td>
<td>14</td>
</tr>
<tr>
<td>3.2.1 Account management</td>
<td>15</td>
</tr>
<tr>
<td>3.2.2 Authentication System</td>
<td>15</td>
</tr>
<tr>
<td>3.3 Shopping Application Module</td>
<td>18</td>
</tr>
<tr>
<td>3.3.1 Fetching encrypted share with AES</td>
<td>19</td>
</tr>
<tr>
<td>3.3.2 Attacker site prototype with setflag</td>
<td>19</td>
</tr>
<tr>
<td>3.4 Database and Swing application</td>
<td>21</td>
</tr>
<tr>
<td>3.4.1 Database</td>
<td>21</td>
</tr>
<tr>
<td>3.4.2 Swing application</td>
<td>22</td>
</tr>
<tr>
<td>CHAPTER</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>------</td>
</tr>
<tr>
<td>3.5 Use Case Diagrams</td>
<td>22</td>
</tr>
<tr>
<td>3.5.1 Use case diagram for Customer Interaction</td>
<td>22</td>
</tr>
<tr>
<td>3.5.2 Use case diagram for BankAdmin Interaction</td>
<td>23</td>
</tr>
<tr>
<td>3.5.3 Use case diagram for Shopping Cart Interaction</td>
<td>23</td>
</tr>
<tr>
<td>3.5.4 Use case diagram for Transaction Process</td>
<td>25</td>
</tr>
<tr>
<td>3.5.5 Use case diagram for Payment</td>
<td>26</td>
</tr>
<tr>
<td>3.6 User Interface</td>
<td>27</td>
</tr>
<tr>
<td>3.6.1 Bank transactions list view</td>
<td>27</td>
</tr>
<tr>
<td>3.6.2 OTP entering Screen for the User</td>
<td>27</td>
</tr>
<tr>
<td>4 TESTING</td>
<td>30</td>
</tr>
<tr>
<td>4.1 Test case: Bill payment page</td>
<td>30</td>
</tr>
<tr>
<td>4.2 Test case: Payment login page for the user</td>
<td>31</td>
</tr>
<tr>
<td>4.3 Test case: User with invalid credentials</td>
<td>31</td>
</tr>
<tr>
<td>4.4 Test case: Generation of OTP when merchant server is verified</td>
<td>32</td>
</tr>
<tr>
<td>4.5 Test case: Conversion of OTP to QR code image</td>
<td>33</td>
</tr>
<tr>
<td>4.6 Test case: Converting QR code into Image shares</td>
<td>33</td>
</tr>
<tr>
<td>4.7 Test case: Received share via email</td>
<td>34</td>
</tr>
<tr>
<td>4.8 Test case: Verification of Registered E-mail and Share sent Email</td>
<td>34</td>
</tr>
<tr>
<td>4.9 Test case: Decryption of two shares to obtain QR code</td>
<td>35</td>
</tr>
<tr>
<td>4.10 Test case: Validation of Merchant server for non attacker case</td>
<td>36</td>
</tr>
<tr>
<td>4.11 Test case: Validation of Merchant server for attacker site case</td>
<td>36</td>
</tr>
<tr>
<td>4.12 Test case: Generated QR code and OTP should be the same</td>
<td>37</td>
</tr>
<tr>
<td>4.13 Test case: Passing authentication if the user entered correct OTP</td>
<td>38</td>
</tr>
<tr>
<td>4.14 Test case: Failed Transaction</td>
<td>39</td>
</tr>
<tr>
<td>5 CONCLUSION AND FUTURE WORK</td>
<td>40</td>
</tr>
<tr>
<td>5.1 Future Work</td>
<td>41</td>
</tr>
<tr>
<td>REFERENCES</td>
<td>41</td>
</tr>
<tr>
<td>APPENDIX A</td>
<td>45</td>
</tr>
</tbody>
</table>
## LIST OF FIGURES

<table>
<thead>
<tr>
<th>FIGURE</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Original Image</td>
</tr>
<tr>
<td>2</td>
<td>Embedded Image</td>
</tr>
<tr>
<td>3</td>
<td>Picture of QR code</td>
</tr>
<tr>
<td>4</td>
<td>Total Architecture Overview</td>
</tr>
<tr>
<td>5</td>
<td>Checkout of list of products</td>
</tr>
<tr>
<td>6</td>
<td>Banking application class diagram</td>
</tr>
<tr>
<td>7</td>
<td>One share of QR code</td>
</tr>
<tr>
<td>8</td>
<td>Carrier image for embedding the share</td>
</tr>
<tr>
<td>9</td>
<td>Carrier Image after embedding the share</td>
</tr>
<tr>
<td>10</td>
<td>Class diagram for Shopping cart module</td>
</tr>
<tr>
<td>11</td>
<td>Setting flag to run non attacker site</td>
</tr>
<tr>
<td>12</td>
<td>OTPMAP mapped to the transaction id</td>
</tr>
<tr>
<td>13</td>
<td>Use case Diagram for Customer Interaction</td>
</tr>
<tr>
<td>14</td>
<td>Use case diagram for BankAdmin Interaction</td>
</tr>
<tr>
<td>15</td>
<td>Use case diagram for Shopping Cart Interaction</td>
</tr>
<tr>
<td>16</td>
<td>Use case diagram for Transaction Process</td>
</tr>
<tr>
<td>17</td>
<td>Use case Diagram for Payment</td>
</tr>
<tr>
<td>18</td>
<td>Bank transactions list of a user</td>
</tr>
<tr>
<td>FIGURE</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td>19</td>
<td>OTP entering Screen for the User</td>
</tr>
<tr>
<td>20</td>
<td>Bill payment page</td>
</tr>
<tr>
<td>21</td>
<td>Payment page with user credentials for the bank</td>
</tr>
<tr>
<td>22</td>
<td>Invalid credentials</td>
</tr>
<tr>
<td>23</td>
<td>Generated OTP in the bank logs</td>
</tr>
<tr>
<td>24</td>
<td>Creation of QR code and shares creation</td>
</tr>
<tr>
<td>25</td>
<td>Received share via email</td>
</tr>
<tr>
<td>26</td>
<td>Verification of Registered E-mail and Share sent Email</td>
</tr>
<tr>
<td>27</td>
<td>Decryption of QR code from shares</td>
</tr>
<tr>
<td>28</td>
<td>Setflag for non attacker site</td>
</tr>
<tr>
<td>29</td>
<td>Setflag for attacker site</td>
</tr>
<tr>
<td>30</td>
<td>Displaying stego image for fake merchant server without decryption key</td>
</tr>
<tr>
<td>31</td>
<td>Text after QR scanner</td>
</tr>
<tr>
<td>32</td>
<td>Generated OTP in logs</td>
</tr>
<tr>
<td>33</td>
<td>Payment Success</td>
</tr>
<tr>
<td>34</td>
<td>Payment failed</td>
</tr>
</tbody>
</table>
CHAPTER 1

BACKGROUND AND RATIONALE

Web application security [1] is an important aspect of modern web design. Security has become an integral part of all computer applications. Various techniques exist to ensure application security. Cryptography [2, 3] and steganography [4, 5] are two central concepts. Cryptography is the process of encrypting the data with various algorithms. Image steganography is the concept of sending secret data in an image (termed as cover image) without changing the cover image. Prior to embedding the secret data in the cover image, the data is encrypted to increase its security levels. Image steganography is a branch of steganography in which many different carrier file formats are used. Visual cryptography [6, 7] is the method of encrypting visual data for authentication purposes in which a secret image is encrypted into shares such that mixing a sufficient number of shares will reveal the hidden image. Steganography is hiding secret information in the cover image. The hidden message must be interpreted only by the sender and the receiver. The existence of the message should be unknown to any intruder, thus avoiding potential security issues [3]. Cryptography protects the message content only, but steganography protects both the message content and the communication parties. The data is concealed within the data bits of the image without distorting the overall appearance of the image [8, 3]. Steganalysis is used to decrypt the data.
1.1 Image Steganography

The purpose of image steganography [4, 5] is to hide data in images. Image steganography mainly has two strategies: spatial domain techniques and transform domain techniques. Transformation or frequency domain techniques are based on the manipulation of the orthogonal transform of the image rather than the image itself [4]. Spatial domain techniques directly deal with the image pixels. The pixel values are manipulated to achieve desired enhancement [4]. Pixel value differencing [9] and Least Significant Bit (LSB) embedding techniques [10] are the most widely-used spatial techniques. In pixel value differencing uses the largest difference value between the other three pixels close to the target pixel, to estimate how many secret bits will be embedded into the pixel. In LSB embedding, some pixels in the image are manipulated so that an attacker [3] cannot recognize the hidden information. After hiding information in a given image, steganography is used to retain the feel of the earlier image that is used to hide the information.

1.1.1 LSB embedding

LSB embedding is a technique in which images are represented as the intensity values of the pixels in each coordinate position [10]. The pixels can be 8-bit, 16-bit, or 24-bit depending on the image's format, size and standard. LSB embedding considers the contribution of the last bit of the image pixel to be less than that of other bits. Therefore, steganography using LSB embedding uses this technique to minimize damage to the image when embedding data. Figures 1 and 2 give example images before and after embedding data in the image, respectively. The bits manipulation in LSB embedding is shown in the following example. By considering three adjacent pixels of an image with RGB encoding:
**Figure 1.** Original Image

**Figure 2.** Embedded Image
Message to be hidden is of 9 bits, which is given by \([10]\) : 101101101 overwriting these 9 bits over the LSB of the 9 bytes above, the result will be:

\[
\begin{align*}
00001101 & \quad 00011100 \quad 11111001 \\
10000110 & \quad 00011111 \quad 11011010 \\
10001111 & \quad 00110000 \quad 11011011
\end{align*}
\]

All the 9 bits are hidden successfully only at cost of changing 9 bits\([8]\).

1.2 Encryption and Decryption Techniques

Encryption is the process of converting information or data into a code, especially to prevent unauthorized access [3]. Many encryption algorithms are available for encryption and decryption. These can be categorized in two groups, symmetric key encryption algorithms [11, 3] and asymmetric key encryption algorithms [3].

In symmetric key encryption, only one key is used for both encryption and decryption, whereas in asymmetric key encryption (also known as public key encryption), two keys are used: public key and private key [3]. The public key is used to encrypt data and the private key to decrypt [3].

1.2.1 Advanced Encryption Standard

Advanced Encryption Standard (AES) is a symmetric-key algorithm [12, 3] with a block size of 128 bits. It supports lengths of 128, 192, and 256 bits. AES is a superior algorithm with advanced features when compared with the data encryption standard (DES) [13]. In AES, there is no feistel network like in standard DES. The
cipher consists of rounds; the number of rounds used depends on the key length: 10 rounds for a 128-bit key, 12 rounds for 192-bit key, and 14 rounds for a 256-bit key. The whole algorithm operates on a 4X4 matrix of bytes. The first round comprises four distinct transformation functions: sub bytes, shift rows, mix columns, and add round key. The final round contains three transformations; the mix columns function is not used in the final round. Each transformation takes one or more 4X4 matrices as input and produces a 4X4 matrix as output. Provided that all four rounds are reversible, decryption does recover the plaintext [3].

1.3 Phishing Process

Phishing is a very common attack in online transactions [14]. Phishing is a method of stealing data in which attackers send fake emails from various websites to collect the users private information [7, 3]. Users will unwittingly enter the password or other private information, thinking it is a legitimate website. Attackers collect this stolen data and use it to gain access to private areas of websites. To tackle this, anti-phishing [14] image based authentication techniques have been proposed [14]. If these use image steganography and visual cryptography, they will be more secure and eliminate phishing.

1.4 Visual Cryptography and QR Code

Quick response (QR) codes are a machine-readable format in which information is embedded in an image comprising black and white pixels [7]. Visual cryptography can be used with QR codes to create shares [7]. Image shares are components of the original image so that any one share (on its own) does not reveal anything about the original image. Each share is later stacked back together to obtain the original
Whenever a user requests authentication, the generated QR will be made into shares. The created shares then overlay with each other to obtain the original QR code for authentication [7]. Figure 3 is an example QR code.

![QR Code](image)

**Figure 3.** Picture of QR code

1.5 Existing Techniques

Application security is a key issue for users [1]. Image steganography [4] and visual cryptography are two significant areas in which the security of visual communication can be improved. Securing visual cryptographic shares using public key encryption involves securing communication using a Ron Rivest, Adi Shamir and Leonard Adleman (RSA) algorithm.

In some techniques, both encryption and steganography are used together. RSA algorithm strengthens the security of visual cryptography [15, 3]. When an image is encrypted using an RSA algorithm, the attacker will be unable to decrypt the information; even creating fake shares will not enable them to access the information [16]. The system provides secret shares, which are robust against multiple attacks, and provides reliable security for handwritten text, images, and printed documents sent over the public network. After images are divided into shares, those shares are
encrypted using an RSA algorithm [15, 16]. To decrypt the shares, the receiver must enter the necessary key using the RSA algorithm. K-means clustering and encryption techniques are also used in this process: the image is divided into clusters using k-means clustering, and the necessary information is concealed in the clusters using RSA public key encryption [17, 8].

Visual cryptography and encryption techniques play vital roles in online fraud prevention [7, 18]. In this technique, a user requests server authentication through a merchant server. The merchant server sends the user id, password and server key to the central server to confirm the verification [7]. The primary server verifies the server and user details, then generates a one-time passcode which is later converted into a QR code for authentication purpose.

Additional cryptography algorithms and visual cryptography techniques have been developed [2, 3]. During a transfer within a network, the shares remain unencrypted. A man-in-the-middle attack can predict the secret message with a single share, but if the attackers can gather all available shares, they could possibly decode the plain text from the image shares using brute force attacks [19]. To avoid this problem, a new approach is proposed in which both encryption and decryption phases will make sure a secure transmission. In the encryption phase, the original image and key are input. After this, the cipher text and image splits are created using new AES novel approach [12].

OTPs and QR codes have become widespread methods to prevent online fraud. Numerous authentication systems are used to ensure that only valid users log in to online bank accounts. In this case, the authority co-ordinates the server and user. The user requests authentication from the server. The server generates a QR code with a random value, transfer information, and required time to transfer [18, 20].
All this information will be sent to the certification authority (CA) [20, 18]. The CA verifies the user information, user-entered OTP, and serial number along with the server-side information; authentication is provided if the information matches.

Some image processing techniques have also introduced signature authentication to this process [21]. The signature image is preprocessed and the shares are created using visual cryptography [21]. Shares are given to bank and the user. Those signature shares are later stacked to reveal the original signature for authentication. A correlation algorithm is applied to test the shares precision and accuracy for authenticity [21].

Security is also a primary concern for online payment applications. Image steganography and visual cryptography are again used to provide authentication. To limit the information that is shared with a merchant server, applications using a certification authority. Certification authority is an authentic third party service designed by bank server to verify authenticity of the user credentials. When payment request is received, the receipt of payment is converted into shares. One share each is given to the merchant and user [20]. The receipt of information, which is retained by the merchant, contains only the card number used for shopping. When user and merchant give the shares to the CA, it sends them to the bank server for verification. The transactions validity is verified by the bank server based on the CA information [18].

Neural networks are used in image steganography to select where the secret image will be hidden. As mentioned previously, the LSB embedding technique is used to hide this information. In LSB embedding, dynamic key cryptography is used to strengthen the system. In this process, the most significant bit of a secret image modifies the LSB of the picture element of carrier image. Carrier image is used to
hide the secret image. This process provides additional security [22]. The dynamic selection of a key is enabled by rotating the key, and each key rotation produces a new key. In this technique, pixel selection of the carrier image and the secret image is performed based on a pseudo-random number, which provides double layer security against the stegano analytic attack [19, 22]. Stegano analytic attack is the reversal process of steganography to decode the hidden information without authorization.

In phishing prevention technique [19], visual cryptography is used to authenticate the user, while at the time of registration the password of the user is concatenated with the random string. The total string is then converted to captcha image, which is divided into shares. One share is given to the user, the other to the server. Both are stacked together to obtain the captcha image [19].

1.6 Motivation

Visual cryptography alone is rarely used in practice as it is limited to dividing an image into shares. As discussed in section 1.5, various techniques have combined visual cryptography and steganography to improve anti-phishing security. Steganography, most commonly LSB embedding, is used to embed an image into a carrier image. However, these methods are vulnerable to statistical attacks and phishing attacks. Stastical attack is exploiting the statistical weaknesses in a targeted algorithm. Therefore, strengthening security remains a priority. Adding AES encryption with a public key cryptosystem to the existing methods is one possibility to eradicate security leaks.
1.7 Project Objective

The project objective is to create an application that uses visual cryptography and AES encryption to prevent online fraud when a scammer attempts to log in through a payment gateway. Implementing above mentioned in two web applications to provide the prototype of actual transactions in the real world. Later, testing the application security against the given requirements.
CHAPTER 2

SYSTEM DESIGN AND ARCHITECTURE

As shown in Figure 4, the security system designed in this project is divided into three modules: a shopping application, bank application and swing java application to merge the shares. Each of the modules is explained with each of their respective functionalities in chapter 3. In this chapter, the system architecture and prerequisites of the system are discussed. Following are the prerequisites for the designed system to start:

- The user should have an account in the bank with updated contact details
- The merchant site and the bank should have a shared public key for encryption and decryption of the share image
- User should have an application to merge the shares and to form the QR code Image
- User should have a QR code scanner application to read the QR code and get the OTP

The system interaction flow is as follows:

- The user adds some items to the cart in the designated merchant site as shown in Figure 5. The figure 5 shows the list of items user is interested in purchasing and their item numbers respectively.
- During checkout the merchant site will navigate to the payment interface to collect the netbanking credentials for payment approval.
When the user enters the credentials and clicks "Pay", the system will pop up a confirmation message to proceed with payment authentication.

If the customer clicks "OK" then the merchant site submits the request to the banking server.
CHAPTER 3

SYSTEM IMPLEMENTATION

In this chapter, the system requirements for the project and implementation of various project modules (Figure 4) are discussed.

3.1 System Requirements

System requirements are the software and minimum hardware requirements to implement the secure transaction using visual cryptography and encryption techniques. The following are the details of software and hardware requirements of the project.

3.1.1 Software Requirements

- Programming Languages: Java, Java Server Pages (JSP)
- Tools: Eclipse IDE version Kepler Service Release 2, Java SE Development Kit 7u80
- Database: MySQL 5.7
- Web server: Apache Tomcat 6.0

3.1.2 Hardware Requirements

- Operating System: Windows 7/8/10
- RAM: 1GB
- Memory: 500GB
3.2 Banking Application module

When the user clicks on the OK button (chapter 2), the merchant server requests authentication from the banking server. Here, the pay option triggers the main authentication part of the application. The banking application module responsible for secure transaction authentication also stores all the customers' data. Various segments are responsible for providing secure user authentication for the user (Figure 6). Figure 6 is a class diagram for banking server module and contains various classes responsible for the authentication system. The different segments (Figure 4) of bank applications are described in the following sections.
3.2.1 Account management

The account management section of the banking application retains user details. It stores the transaction data of each user. It is connected to the database with java database driver connection (JDBC) through a java environment in which all the back-end tables are accessed and stored and inputs are given from the user interface like adding users, creating new accounts, and providing user-requested transaction data.

3.2.2 Authentication System

When the payment request from the merchant site is made it will be served by the LoginController class (Figure 6) section, the other option sections are for normal user log in and administrator log in actions of the banking site. Authentication system contains various java class files which will be required for authentication process. Figure 6 is the class diagram displaying all the classes in the banking application.

**OTP Generator:** The OTP is a one-time pass code generated for each transaction. The OTP generator is responsible for random number generation of six digits in length. When the user requests a payment using their credentials, the request is verified by the bank. The bank server checks the user data in the database. If the user is valid, it generates a random string of numbers using a random function. It also generates a transaction number to track each transaction. OTPMAP is a table in banking database. The bank server then stores the generated OTP string in the OTPMAP table against the transaction ID for OTP verification and passes the OTP to the QR code generator.

**QR code Creation:** QR code is created using the generated random string by the OTP generator class. Fig 3 shows the example QR code. QR code generator class
(Figure 6) is responsible for QR code generation. Zxing [23] library is a Java [24] library which is uses to create QR code, bar codes etc.

**Visual Cryptography on QR code image:** Visual cryptographer class (Figure 6) in banking module calls all the functions to perform different operations required for authentication purpose. First, the generated QR code image is collected to apply visual cryptography techniques. (2,2) visual cryptographic scheme is used to divide the image into two shares. White and black pixels are separated into two different images called shares. In this technique the image will be divided into two parts one is a key image (transparent image) and the other one is a cipher image. Taken separately, they appear to be random noise images, but combining them reveals an image. XOR algorithm is used to create the share images.

**XOR based Visual Cryptography** XOR based visual cryptography is used to create the shares of QR code image.

The process of XOR visual cryptography is as follows:

- Create an image of random pixels the same size and shape as the original image
- Create a random second image whose pixels are the exclusive-or (XOR) of the first image and the original image which will be a random image.
- Iterate the pixels of the key image (share 1) one by one and perform the XOR operation of the pixel values in the original image.
  - If the key pixel is black, flip the color of the original image and store in cipher at identical location
  - If the key pixel is white, set the cipher pixel to the same color as the original image
• Using this XOR operation the cipher image (share 2) is generated

• After forming the key and cipher images, both are magnified by two times by rearranging the pixel values

• The image pixels will be divided by four subpixels
  
  – if pixel is black then set 0,0 and 1,1 as black and 0,1 and 1,0 as white pixels
  
  – If pixel is white then set 0,0 and 1,1 as white and 0,1 and 1,0 as black pixels

After two shares are created, one share is sent to the user by email. Email processor class will be sending the share to the user with the registered email.

**AES encryption on Image Share and Steganography:** Steganography is hiding the image into a different carrier image such that only intended user can access. Output of the visual cryptography (2,2) scheme is two shares, one of the shares is already being sent to the user’s email. The second share is encrypted using AES encryption with a public key selected by banking server. The secret image (share 1) is converted to array of bytes using `getImageBytes` method. The converted image bytes is encrypted using AES encryption technique. The only request with the public key can be able to pass through the authentication process. After the share is encrypted using AES encryption LSB embedding is applied on the share.

For LSB embedding, a carrier image is required to hide the share. The carrier image should be greater than 250x250 pixels which is a limitation to select the carrier image. Figure 7, Figure 8 represents the share image, carrier image respectively. To embed the share into carrier image LSB embedding technique is used. After embedding all the bytes in carrier image, encrypted share 1 length also embedded in
Figure 7. One share of QR code

Figure 8. Carrier image for embedding the share

The size of the image is also a key factor in image encryption techniques. Initially, the size of the carrier image is 220 KB (Figure 8). However, after embedding, it is 276 KB, 56 KB larger (Figure 9). The size of the new image is acceptable because anything larger would be notable and therefore vulnerable to attacks during transfer.

Figure 9. Carrier Image after embedding the share

3.3 Shopping Application Module

The shopping module performs various functions like selecting the items, user login, user registration and storing the list of items in the cart. The shopping cart module
is responsible for decrypting the given share with the AES key provided by the bank server. The shopping application class diagram is shown in Figure 10.

3.3.1 Fetching encrypted share with AES

When the merchant server is requested for payment with valid AES, bank server fetches the encrypted carrier image from the carrier image. The steganograph class (Figure 10) decrypts the given stego image using the shared AES key. After fetching the encrypted image, the encrypted share 1 (merchant share) is retrieved from the carrier image as follows.

- The carrier image is converted to RGB pixel byte values
- From the last four bytes the size of the encrypted share is read
- The carrier image pixels are iterated to extract the encrypted image contents until encrypted image length is reached

After fetching the encrypted share, AES encryption is applied on the encrypted share 1 to obtain the original share created by the bank server. The shared AES key is required to decrypt the hidden share which can only be shared by the bank server. The decrypted share will be displayed in the web page at the time of transaction.

3.3.2 Attacker site prototype with setflag

The phishing prevention class (Figure 10) ensures authentic communication between the bank and merchant sites. In the phishing prevention system, the run attacker site flag (Figure 11) evaluates the behavior of the site and determines whether the site is authentic or fake. A "true" flag value denotes an attacker site, and "false" an authentic site. However, still needs to be verified. At this stage in the process,
the merchant server requires the AES key. Only with the authentic security key can the merchant complete the payment process. Figure 11 shows a flag set to 0 for a regular phishing-free transaction. If the flag is set to 1, an attacker site scenario will be executed and the user will be unable to complete the transaction as the attacker site has been identified.

Figure 10. Class diagram for Shopping cart module

Figure 11. Setting flag to run non attacker site
3.4 Database and Swing application

3.4.1 Database:

In this project, both shopping cart database and vcnetbank databases are available. The shopping cart database stores the list of products, customer user name and login information. The vcnetbank database also retains tables like OTPMAP, customer which are part of the project. Among all the tables, the OTPMAP and customer tables are mostly frequently required. The customer table stores all customer details required for bank transactions. Whenever a customer requests access for an online transaction, the bank server checks the details in the customer table and generates the OTP. The OTPMAP table stores the OTP generated with reference to the transaction ID (Figure 12). Figure 12 displays the OTP mapping to each transaction id.

**Figure 12.** OTPMAP mapped to the transaction id
3.4.2 Swing application

The swing application is created using java, which is used to stack both image shares to obtain the QR code. When the user obtains both shares (the one displayed on the website and the one emailed to the user), the swing application merges both shares and obtains the QR code. It converts both share 1 and share 2 into byte arrays of pixels. The decryption is performed in blocks of four pixels at a time. If diagonal pixels are white or black, then all four pixels will be set to black or white, respectively. By this method, the XOR operation is applied to the magnified images.

3.5 Use Case Diagrams

3.5.1 Use case diagram for Customer Interaction

This use case as shown in Fig 13 demonstrates user interaction with the merchant and bank sites. Numerous interactions are possible for the customer, as follows.

- The customer will be able to log in and log out of the bank module
- When customer tries to login, the user credentials are verified with the existing credentials in the banking database.
- The customer can view account details and update contact details.
- Customer can view all previous transactions on the view transactions page in the bank application.
- When the customer requests access, the bank application checks the banking database with existing user credentials.
3.5.2 Use case diagram for BankAdmin Interaction

The use case diagram for BankAdmin interaction (Figure 14) details how banking admin staff interact with the databases and functionalities. The flow of interaction is as follows.

- Bank admin staff log into the system using admin credentials
- Bank admin are authorized to create and manage user accounts. Account creation can only be performed by bank admin staff
- Bank admin view customer transactions

3.5.3 Use case diagram for Shopping Cart Interaction

Figure 15 shows the use case diagram for shopping cart interactions. The shopping cart contains various functionalities such as user account creation, user credential
Figure 14. Use case diagram for BankAdmin Interaction

verification, item selection, cart stocking, and purchasing. These operations are detailed below:

- Login User: In this use case, the user will log in to the shopping cart application to select items to purchase. At this stage, the shopping cart verifies user data with the database.

- View Cart Items/Make purchase: The user selects which items to buy and checks them out to pay the bill.

- Check out items: After checking out items, the application progresses to the "confirm cart" items page

- Register new user: New users can also register on the shopping cart application registration page
3.5.4 Use case diagram for Transaction Process

In the transaction process, the shopping cart, user, and bank are all actors. Following is a list of steps in which these actors are involved during the transaction, as shown in Figure 16

- When the user select list of items for purchasing, the list is verified with the existing items in the shopping cart database

- When the user clicks on "Make Purchase", the check out page is displayed

- If the user is checking out, the payment page loads

- When the payment page loads, eventually the bank application is displayed. This requests the user’s credentials
• This process includes verifying the user authentication with bank. If Customers are verified in the database check, the OTP is generated

• Merchant share is displayed after the visual cryptography process on QR code image, if the given credentials are valid

![Use case diagram for Transaction Process](image)

**Figure 16.** Use case diagram for Transaction Process

3.5.5 Use case diagram for Payment

To ensure successful transaction, the OTP must be correct. The OTP is obtained after the user scans the QR code. The following steps are involved in this transaction, as shown in Figure 17.
To complete the purchase, the customer must log in to their bank

Customer requests the bank via merchant server for payment authentication

When this request is made, merchant forwards the request to the bank admin

The bank admin verifies the request, display share 1, share 2 will be emailed to the user

The swing application collects both shares to create the QR code

Customer will scan the QR code and enter the OTP to verify the authentication

The "successful payment" screen will be displayed if the customer enters the correct OTP. If they do not, payment will fail.

3.6 User Interface

3.6.1 Bank transactions list view

The list of banking transactions of user will be shown as Figure 18. In the view transactions, the customer can select the account number to view the debit, credit transactions of the particular account. The transaction id, date of transaction, amount debited and the transaction type is shown as Figure 18.

3.6.2 OTP entering Screen for the User

In the OTP input screen, the customer enters the OTP generated by the scanned QR code (Figure 19). In Figure 19, the figure represents the share 1 that is created from the QR code and dispalyed in the merchant server page. The user has to download the figure and merge it with user share to obtain the QR code.
Figure 17. Use case Diagram for Payment

Figure 18. Bank transactions list of a user
Figure 19. OTP entering Screen for the User
CHAPTER 4

TESTING

Testing is required to make sure the system is functioning as per the given requirements of the project. To verify and validate the secure payment authentication system, the positive and negative test scenarios are executed.

4.1 Test case: Bill payment page

The test case 4.1 is an example of bill payment page as shown in Figure 20. The bill payment page is displayed when the user clicks the check out (Figure 5) button after selecting the list of items. The bill payment page as shown in Figure 20 displays serial number, item name, quantity, rate, total for each item selected by user. Serial number is product identification number. The item name is the name of the item selected by the user to purchase, quantity is the how many items user wants to purchase, rate is price of each item and total is the price of all the items selected by the user. The total amount displayed below the table in Figure 20 shows the total sum of all the items.

![Bill payment page](image)

Figure 20. Bill payment page
4.2 Test case: Payment login page for the user

The test case 4.2 is an example of payment login page for payment authentication as shown in Figure 21. When a user clicks on the payment button as show in the Figure 5 the payment login page is displayed. The payment login page displays the user name, password and amount to be paid. In this page user has to enter the banking user name and password as shown in Figure 21. Banking user name and password are the credentials created by the user while registering in the bank.

![Payment page with user credentials for the bank](image)

**Figure 21.** Payment page with user credentials for the bank

4.3 Test case: User with invalid credentials

The test case 4.3 is an example of failure of authentication when user enters invalid credentials as shown in Figure 22. When user enters the user id and password, the bank will compare the user id and password with the existing user credentials in the database. User in the payment page has to enter the correct login credentials. The user will not be allowed to complete the transaction with invalid credentials. In the Figure 22 the "invalid credentials" error is displayed in red color in case of wrong user id or password.
4.4 Test case: Generation of OTP when merchant server is verified

The test case 4.4 is an example of one time passcode generation when merchant server is verified as shown in Figure 23. When user requests the access from the bank server along with merchant server URL, it will first go to the merchant server. When merchant server is authenticated with bank server the bank server will generate an OTP. Generated OTP will be stored in vcnetbank.log. The vcnetbank.log is a log file storage for banking server. It stores all the information about user validation and OTP at the time of transaction.
4.5 Test case: Conversion of OTP to QR code image

The test case 4.5 is an example of converting the one time pass code to QR code image as shown in Figure 24. When the merchant server requests for authentication from bank server, bank server will generate the OTP. OTP is one time passcode string generated by a random number generator. Generated OTP will be converted to QR code image stored in a location specified by the bank server as shown in Figure 24.

![Figure 24. Creation of QR code and shares creation](image)

4.6 Test case: Converting QR code into Image shares

The test case 4.6 is an example of converting the QR image into different share images using visual cryptography as shown in Figure 24. Share is a secret image created after visual cryptography process. After the creation of QR image, the QR image is converted into two different images. Those two different images are called as secret shares. The generated shares will be stored in a location specified by bank server as shown in Figure 24. One share will be given to the server; another share will be sent to the user via email as shown in Figure 25.
4.7 Test case: Received share via email

The test case 4.7 is an example of receiving one of the shares via an email from the bank server as shown in Figure 25. When user clicks the "Pay" in Figure 21 it will generate two share image from QR code after visual cryptography process (Figure 24). One of the shares created by the visual cryptography process will be sent to the user’s registered email at the bank as shown in Figure 25.

![Received share via email](image_url)

**Figure 25.** Received share via email

4.8 Test case: Verification of Registered E-mail and Share sent Email

The test case 4.8 is an example of verifying registered email with the email to which the secret share is being sent as shown in Figure 26. User share will be sent to the registered email in the bank database given by the user. In Figure 26 it is shown that the bank account number of the user, user’s email id is mapped in the database.
4.9 Test case: Decryption of two shares to obtain QR code

The test case 4.9 is an example of obtaining the QR image after collecting both shares from user’s email and downloaded share from the bank server web page as shown in Figure 27. Both merchant share and user share are stacked together to obtain the QR code. Once the QR code is obtained user will have to scan the QR code with QR code scanner. The obtained OTP will be entered in the payment gateway. If it is success, user will be able to complete the transaction successfully.
4.10 Test case: Validation of Merchant server for non attacker case

![Image of Eclipse IDE]

**Figure 28.** Setflag for non attacker site

The test case 4.10 is an example of validation of merchant server by setting the "runattackersite" flag as shown in Figure 28. When user request access to bank server during the transaction from a merchant server, it will request the access from bank server to the data. Bank server allows the merchant server with only proper public key access. If the setflag is set to 0, the merchant server with valid AES key will be allowed by bankserver. Merchant server share is displayed as shown in Figure 19.

4.11 Test case: Validation of Merchant server for attacker site case

The test case 4.11 is an example of when a fake server URL is asking the user to enter credentials to the server, as the bank server having the public key encryption, the unauthorized merchant server can not decrypt the share without public key.

To run the prototype of an attacker, first the "runattackersite" flag should be set to 1 as show in Figure 29. Figure 30 shows where the merchant server failed to access the encrypted share, showing a carrier image (Figure 9) in which the encrypted
4.12 Test case: Generated QR code and OTP should be the same

When the user is verified, the bank server generates the OTP. The generated OTP is stored in vcnetbank.log file on the bank server. After the swing application merges the shares, the user must scan the QR code with a mobile app (Figure 31). The OTP stored in the vcnetbank.log file and the scanned QR code should match (Figure 32).
Vcnetbank.log is file maintained by the bank server to record the logs of transactions. After merging the shares in swing application, user has scan the QR code with the mobile app. The OTP stored in the vcnetbank.log file and the scanned QR code should be the same.

4.13 Test case: Passing authentication if the user entered correct OTP

After scanning the QR, the user enters the OTP in the relevant field to complete the authentication. If the OTP matches with the one in the database, the transaction is completed and transaction ID is displayed (Figure 33).
4.14 Test case: Failed Transaction

When the merchant server is failed to decrypt the share as shown in (Figure 30) and tries to enter the wrong OTP, the transaction will be failed (Figure 34). Error warning message is displayed for user that phishing attempt is made as shown in Figure 34.
CHAPTER 5

CONCLUSION AND FUTURE WORK

Security is a major concern of web users, particularly when using banking applications. Every time user requests a transaction from their bank, the merchant server must be verified to eliminate any chance of fraud. The fraudster may create a fake merchant server that asks the user for their banking credentials. There have been many cases of users falling for a fraudsters web application and entering their credentials. However, several fraud prevention tools that provide secure transactions are available.

Numerous security mechanisms are being developed in the area of visual cryptography. The secure transaction using authentication process helps users to identify legitimate merchant servers with the help of visual cryptography techniques. In these systems, two different images shares are created and shared with the merchant server and user. During this process, the bank server is responsible for authenticating the user and merchant server. The bank server collects the generated OTP from the user, which is later compared with the previously generated OTP in the bank server database. The report above describes how this process has been successfully implemented in the proposed system. The proposed system has advantages over existing techniques as they lack public key encryption mechanism between the bank and merchant. Applying such encryption mechanisms in the communication channel helps to avoid the middle man attack between the bank and merchant. Along with the secure communication, attacker site prototype is also successfully implemented as an example of phishing prevention.

The secure transaction processing system presented here enables fully secure
banking transactions. The aim of this application is to provide a secure login to the bank during the transaction involving merchant application.

5.1 Future Work

In future works, this application could be extended by implementing dynamic LSB steganography techniques. In addition, developing machine learning features to the system and determining a way to select optimum points to hide images will further improve the system. Currently, when shares are merged, there is no authentication testing mechanism; authentication testing would improve the quality of share overlapping, which can be achieved through correlation algorithms. Adding correlation algorithms in future will help verify the authenticity of stacked shares, and will improve the results.

REFERENCES


[4] Bin Li, Junhui He, Jiwu Huang, and Yun Qing Shi. A survey on image steganography and steganalysis.


[18] Young Sil Lee, Nack Hyun Kim, Hyotaek Lim, HeungKuk Jo, and Hoon Jae Lee. Online banking authentication system using mobile-otp with qr-code. In


APPENDIX A

CODE SNIPPETS FOR IMPORTANT PROJECT MODULES

Steganograph.java in Shopping cart module:

```java
public static BufferedImage reveal(BufferedImage carrierDir, String password) throws IOException, NoSuchAlgorithmException, NoSuchProviderException, NoSuchPaddingException, InvalidKeyException, IllegalArgumentException, ShortBufferException, IllegalBlockSizeException, BadPaddingException {
    try {
        Field field = Class.forName("javax.crypto.JceSecurity").
        getDeclaredField("isRestricted");
        field.setAccessible(true);
        field.set(null, java.lang.Boolean.FALSE);
    } catch (Exception ex) {
        logger.error("Exception:"+ex.getMessage());
    }

    byte payload[] = null;
    byte[] tmp = null;
    int payloadRemaining = 0;
    int payloadSize = 0;
    int msgLen = 0;
    int bytesToDecodeFromCarrier = 0;
    ArrayList<byte[]> payloadData = new ArrayList<byte[]>();

    offset = 0;
    carrier = convertImageToRGBPixels(carrierDir);

    bytesToDecodeFromCarrier = carrier.length / 2 - 4;

    tmp = decode(carrier, 4); // extracting the payload size
    payloadSize = toInteger(tmp);
    payloadRemaining = payloadSize;
    bytesToDecodeFromCarrier -= 4;

    if (payloadRemaining > bytesToDecodeFromCarrier) {
        payload = decode(carrier, bytesToDecodeFromCarrier);
        payloadRemaining = payloadRemaining - bytesToDecodeFromCarrier;
    } else {
        payload = decode(carrier, payloadRemaining);
        payloadRemaining = payloadRemaining - payloadRemaining;
    }
```

payloadData.add(payload);

if (payloadRemaining > 0) {
    throw new IllegalArgumentException("Some Stego Files missing!");
}
if (!payloadData.isEmpty()) {
    byte[] secretData = new byte[payloadSize];
    byte[] secretFile;
    int ptr = 0;
    for (int i = 0; i < payloadData.size(); i++) {
        byte[] tmpArray = payloadData.get(i);
        for (int j = 0; j < tmpArray.length; j++, ptr++) {
            secretData[ptr] = tmpArray[j];
        }
    }
    MessageDigest md = MessageDigest.getInstance("SHA-256");
    md.reset();
    md.update(password.getBytes());
    secretData = decryptPayload(secretData, md.digest());
    payloadSize = secretData.length;

    secretFile = new byte[payloadSize];
    for (int i = 0; i < payloadSize - msgLen; i++) {
        secretFile[i] = secretData[i];
    }
    return getImage(secretFile);
}
return null;

VisualCryptographer.java

//Generate 6 digit OTP
String otpString = OTPGenerator.generateOTP(customerId);

//Store the OTP against transaction ID for verification
TransactionBL transactionBL = new TransactionBL();
transactionBL.insertOTP(otpString, transactionId);

//Generate the QR code image and embed the OTP in it
BufferedImage bufferedImage = null;
try {
    bufferedImage = QRCodeGenerator.createQRImage(otpString);
} catch (WriterException e) {
    e.printStackTrace();
}
BufferedImage[] shares = VisualCryptographer
.createShares(bufferedImage);
BufferedImage carrierImage = ImageIO.read(new File(CARRIER_IMAGE));
BufferedImage encryptedShare = Steganograph.hide(carrierImage, 
shares[0], AES_KEY);
ImageIO.write(encryptedShare, "PNG", new File( 
"\Test\Enc.png");
sendEmail(customerId, shares[1], transactionId);
request.setAttribute("Share1", encryptedShare);

Steganography.java

} public static BufferedImage hide(BufferedImage carrierDir, BufferedImage secretFile, 
String publicKey) throws IOException, NoSuchAlgorithmException, 
NoSuchProviderException, NoSuchPaddingException, 
InvalidKeyException, IllegalStateException, ShortBufferException, 
IllegalBlockSizeException, BadPaddingException {
try {
Field field = Class.forName("javax.crypto.JceSecurity").
.getDeclaredField("isRestricted");
field.setAccessible(true);
field.set(null, java.lang.Boolean.FALSE);
} catch (Exception ex) {
logger.error("Exception", ex);
}
byte[] payload = getImageBytes(secretFile);
int payloadSize = payload.length;
int freeSpaceInCarrier = 0;
int _bytesWritten;
int payloadOffset = 0;
MessageDigest md = MessageDigest.getInstance("SHA-256");
payload = encryptPayload(payload, md.digest(publicKey.getBytes()));
payloadSize = payload.length;
offset = 0;
_bytesWritten = 0;
carrier = convertImageToRGBPixels(carrierDir);
freeSpaceInCarrier = carrier.length / 2;
freeSpaceInCarrier -= encode(getBytes(payloadSize), 4, 0);

if (freeSpaceInCarrier < payloadSize) {
_bytesWritten = encode(payload, freeSpaceInCarrier, payloadOffset);
} else {
_bytesWritten = encode(payload, payloadSize, payloadOffset);
}
freeSpaceInCarrier -= _bytesWritten;
payloadSize -= _bytesWritten;
payloadOffset += _bytesWritten;

if (payloadSize > 0) {
throw new IllegalArgumentException("Not enough cover images");
}
return convertRGBPixelsToImage(carrier);
private static int encode(byte[] payload, int bytesToWrite, int payloadOffset) {
    int bytesWritten = 0;
    for (int i = 0; i < bytesToWrite; i++, payloadOffset++) {
        int payloadByte = payload[payloadOffset];
        bytesWritten++;
        for (int bit = 7; bit >= 0; --bit, ++offset) {
            // assign an integer to b, shifted by bit spaces AND 1
            // a single bit of the current byte
            int b = (payloadByte >>> bit) & 1;
            // assign the bit by taking[(previous byte value) AND 0xfe]
            // or bit to
            try {
                carrier[offset] = (byte) ((carrier[offset] & 0xFE) | b);
            } catch (ArrayIndexOutOfBoundsException aiobe) {
                // Handle exception
            }
        }
    }
    return bytesWritten;
}

public BufferedImage reveal(BufferedImage carrierDir, char[] password)
    throws IOException, NoSuchAlgorithmException,
        NoSuchProviderException, NoSuchPaddingException,
        InvalidKeyException, IllegalStateException, ShortBufferException,
        IllegalBlockSizeException, BadPaddingException {
    byte payload[] = null;
    byte[] tmp = null;
    int payloadRemaining = 0;
    int payloadSize = 0;
    int msgLen = 0;
    int bytesToDecodeFromCarrier = 0;
    ArrayList<byte[]> payloadData = new ArrayList<byte[]>();
    offset = 0;
    carrier = convertImageToRGBPixels(carrierDir);
    bytesToDecodeFromCarrier = carrier.length / 8 - 4;
    tmp = decode(carrier, 4); // extracting the payload size
    payloadSize = toInteger(tmp);
    payloadRemaining = payloadSize;
    bytesToDecodeFromCarrier -= 4;
    if (payloadRemaining > bytesToDecodeFromCarrier) {
        payload = decode(carrier, bytesToDecodeFromCarrier);
payloadRemaining = payloadRemaining - bytesToDecodeFromCarrier;
} else {
    payload = decode(carrier, payloadRemaining);
    payloadRemaining = payloadRemaining - payloadRemaining;
}

payloadData.add(payload);

if (payloadRemaining > 0) {
    throw new IllegalArgumentException("Some Stego Files missing!");
}

if (!payloadData.isEmpty()) {
    byte[] secretData = new byte[payloadSize];
    byte[] secretFile;
    int ptr = 0;
    for (int i = 0; i < payloadData.size(); i++) {
        byte[] tmpArray = payloadData.get(i);
        for (int j = 0; j < tmpArray.length; j++, ptr++) {
            secretData[ptr] = tmpArray[j];
        }
    }

    MessageDigest md = MessageDigest.getInstance("SHA-256");
    md.reset();
    md.update(new String(password).getBytes());
    secretData = decryptPayload(secretData, md.digest());
    payloadSize = secretData.length;

    secretFile = new byte[payloadSize];
    for (int i = 0; i < payloadSize - msgLen; i++) {
        secretFile[i] = secretData[i];
    }
    return getImage(secretFile);
}
return null;

private byte[] decode(byte[] carrier, int bytesToRead) {
    byte[] _decode = new byte[bytesToRead];
    for (int i = 0; i < _decode.length; ++i) {
        for (int bit = 0; bit < 8; ++bit, ++offset) {
            try {
                _decode[i] = (byte) ((_decode[i] << 1) | (carrier[offset] & 1));
            } catch (ArrayIndexOutOfBoundsException aiobe) {
            }
        }
    }
    return _decode;
}
private static BufferedImage convertRGBPixelsToImage(byte[] carrier) {
    ColorSpace cs = ColorSpace.getInstance(ColorSpace.CS_sRGB);
    int[] nBits = { 8, 8, 8 };  // assuming 8-bit band offsets
    int[] bOffs = { 2, 1, 0 };  // band offsets r g b
    int pixelStride = 3;  // assuming r, g, b, skip, r, g, b, skip...
    ColorModel colorModel = new ComponentColorModel(cs, nBits, false, false, Transparency.OPAQUE, DataBuffer.TYPE_BYTE);
    WritableRaster raster = Raster.createInterleavedRaster(new DataBufferByte(carrier, carrier.length), width, height, width * 3, pixelStride, bOffs, null);
    return new BufferedImage(colorModel, raster, false, null);
}

private static byte[] convertImageToRGBPixels(BufferedImage image) throws IOException {
    width = image.getWidth();
    height = image.getHeight();
    BufferedImage clone = new BufferedImage(width, height, BufferedImage.TYPE_3BYTE_BGR);
    Graphics2D graphics = clone.createGraphics();
    graphics.drawRenderedImage(image, null);
    graphics.dispose();
    image.flush();
    WritableRaster raster = clone.getRaster();
    DataBufferByte buff = (DataBufferByte) raster.getDataBuffer();
    return buff.getData();
}

private static byte[] encryptPayload(byte[] payload, byte[] password) throws NoSuchAlgorithmException, NoSuchProviderException, NoSuchPaddingException, InvalidKeyException, IllegalBlockSizeException, BadPaddingException {
    SecretKeySpec key = new SecretKeySpec(password, "AES");
    Cipher cipher = Cipher.getInstance("AES");
    cipher.init(Cipher.ENCRYPT_MODE, key);
    byte[] cipherText = new byte[cipher.getOutputSize(payload.length)];
    int ctLength = cipher.update(payload, 0, payload.length, cipherText, 0);
    ctLength += cipher.doFinal(cipherText, ctLength);
    return cipherText;
}

private byte[] decryptPayload(byte[] payload, byte[] password) throws NoSuchAlgorithmException, NoSuchProviderException, NoSuchPaddingException, InvalidKeyException, IllegalStateException, ShortBufferException, IllegalBlockSizeException, BadPaddingException {
SecretKeySpec key = new SecretKeySpec(password, "AES");
Cipher cipher = Cipher.getInstance("AES");
cipher.init(Cipher.DECRYPT_MODE, key);
byte[] plainText = new byte[cipher.getOutputSize(payload.length)];
int ptLength = cipher.update(payload, 0, payload.length, plainText, 0);
ptLength += cipher.doFinal(plainText, ptLength);
return plainText;

private int toInteger(byte[] b) {
    return (b[0] << 24 | (b[1] & 0xFF) << 16 | (b[2] & 0xFF) << 8 | (b[3] & 0xFF));
}

private static byte[] getImageBytes(BufferedImage originalImage) {
    ByteArrayOutputStream baos = new ByteArrayOutputStream();
    byte[] imageInByte = null;
    try {
        ImageIO.write(originalImage, "png", baos);
        baos.flush();
        imageInByte = baos.toByteArray();
        baos.close();
    } catch (IOException e) {
        logger.error("Exception", e);
    }
    return imageInByte;
}

private static BufferedImage getImage(byte[] imageInByte) {
    InputStream in = new ByteArrayInputStream(imageInByte);
    BufferedImage bImageFromConvert = null;
    try {
        bImageFromConvert = ImageIO.read(in);
    } catch (IOException e) {
        logger.error("Exception", e);
    }
    return bImageFromConvert;
}

private static byte[] getBytes(int i) {
    return (new byte[] { (byte) (i >> 24), (byte) (i >> 16), (byte) (i >> 8), (byte) i });
}