National College of Ireland

Using cryptography and image steganography to securely transfer data. (Configuration Manual)

MSc Research Project
M. Sc. In Cybersecurity

Yash Lathigara
Student ID: 20182384

School of Computing
National College of Ireland

Supervisor: Professor Imran Khan

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Yash Lathigara		
Student ID:	20182384		
Programme:	M. Sc. In Cybersecurity	**Year:**	2021 - 2022
Module:	M. Sc. Research Project		
Lecturer:	Professor Imran Khan		
Submission Due Date:	16th December 2021		
Project Title:	Using cryptography and image steganography to securely transfer data.		
Word Count:	2517	**Page Count:**	13

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Yash Mahesh Lathigara

Date: 15th December 2021

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Using cryptography and image steganography to securely transfer data. (Configuration Manual)

Yash Lathigara
Student ID: 20182384

Introduction

This handbook was created to outline the methods for executing the research project and to provide the setup of the equipment used to produce and run the models. The stages involve downloading and installing the essential software and packages, as well as the minimal setup required for the project to work well.

Requirements - Software

- OS: Any of the Windows 7, 8, 10 or Linux or Mac
- Front End : JAVA (jdk1.5 or Above Required)
- Tools : JCreator

Requirements - Hardware

- Pentium IV processor or Higher
- Ram: Min 256 MB RAM
- Hard Disk: Min 10GB OF HADDISK

Important coding concepts

- Bytes, Bit Operations, Images, ImageIO, Graphics2D, Raster, DataBufferByte etc.

These are the primary concepts that help to comprehend Steganography.

1) Bytes:

Most programs use this as their basic data source whereas many developers will never use bytes in their source codes. A byte consists of 8 bits either 1s or 0s. those 8 bits (1s and 0s) have decimal value. To get that we can convert the binary number to decimal number easily.

Positional value: 128, 64, 32, 16, 8, 4, 2, 1.

Examples:

00000100 = 4

00000110 = 6

00000101 = 5

00001111 = 15

And so on…

In Java, a byte can be converted from an int with the help of simple concept of casting:

BYTE A = (BYTE) 5;

Most Java classes offer a method for returning an object's byte [], either as a part of the object or the complete entity.

Example of a String Class:

CODE

```
String Y = "Will";
Byte [] A = Y.getBytes();
String Y = "Will";
Byte [] A = Y.getBytes();
```

Where A[0] now has the value for 'W' 87(ascii). It is a byte but seems to be an int if the code was printed onto a display. This byte stores the value of “Will” in bits value.

2) Bit:

Most users of the system and programmers have heard of or used the following basic operations:

AND:

The AND operator joins two bytes. The same rules apply to bit operations as they do to true / false values, where 1 equals true and 0 equals false. If both bytes have a 1 in the same place, the result for that position is 1, otherwise it is 0.

Example:

01000101 = 69
01000101 = 101
01010111 = 87
Byte A = 101 & 87 (AND)

This is gives: 01000101 i.e. 69

OR:

The bit of the OR operator will add 2 bytes. Same as AND, where 1 is true and 0 is false, if one of the bits within the location is a 1, the outcome is a 1.

Example:
01010111 = 87
01100101 = 101
01110111 = 119

Byte A = 87 | 101 (OR)
This gives : 01110111 (119)

On top of these critical activities, get ready to move bits:

LEFT SHIFT:

This to remember when performing left shift is that is the 1st bit isn't 1. One left shift will impact 2 values.

What happens in this situation is that: 0 is added on the right side of the bit sequence, at this time the farthest left most bit is discarded, resulting in a new set of byte. Furthermore, when performing shift in Java, several places to shift must be mentioned. If the value is greater than 1, the process is essentially repeated numerous times, beginning with the preceding shift's result. Any value that is shifted 8 times in this manner will become 0.

Examples:

01010111 (87) when performed Single left shift, we get 10101110 (174)
01010111 (87) when performed Double left shift, we get 01011100 (92)

CODE

BYTE A1 = 87 << 1;
= 10101110

BYTE A2 = 87 << 2;
= 01011100

RIGHT SHIFT:

This is a right shift which is the exact opposite of left shift technique i.e., we add 0 bit to the left most bit and the right most bit is eliminated. Resulting in a new set of byte,
\

Examples:

01010111 (87) when performed Single right shift, we get 00101011 (43)
01010111 (87) when performed Double right shift, we get 00010101 (21)

CODE

BYTE A1 = 87 >>> 1;
We get 00101011
BYTE A2 = 87 >>> 2;
We get 00010101

bit and byte operators are used to successfully create the proposed system.

3) Buffered Image:

A buffered Image is a class that will be used with images. They introduced the ImageIO class in Java 1.5.0, along with methods for accessing the picture's raster and buffer, making image manipulation considerably easier. Buffered image may be used to create new images as follows:

```
BufferedImage img = new BufferedImage(int, int, int);
File file = new File (String);
BufferedImage img = ImageIO.read(file);
```

4) ImageIO:

This class will assist us in doing image input and output operations. This class has many more functionalities, but for the sake of this application, just the read () and write () methods will be utilized.

5) Graphics2D:

A long-standing Java class that provides access to some of the more advanced aspects of graphics/images. Allows you to create customizable sections in a new or existing image. As well as allowing access to the image's renderable section. This class, like others, allows for an easy transition from image space to user space, which is essential for editing or reading individual bytes in a photo.

6) WritableRaster:

We utilize this to generate an image pixel per pixel, which is required when we wish to access a picture's byte (made of pixels). WritableRaster is a Raster subclass that provides methods for gaining more direct access to an image's buffer.

7) To add user space

CODE

```
private BufferedImage user_space(BufferedImage image)
```

```
{
BufferedImage new_img  = new BufferedImage(image.getWidth(),
image.getHeight(), BufferedImage.TYPE_3BYTE_BGR);
Graphics2D      graphics   = new_img.createGraphics();
graphics.drawRenderedImage (image, null);
graphics.dispose();
return new_img;
}
```

- To enter user space, a new image of the same dimensions as the source is created, together with a graphics section.
- The previous image is then rendered/drawn onto the new image.
- As an extra memory advantage, the new image's resources are freed.

The new image is now entirely in user space, which implies that all of the data has been generated and can thus be modified within Java. There are issues with directly altering a photograph; changes are not always executed. It is also advised that you generate this user space as a new clone of the input image, guaranteeing there's no memory sharing between both the source and user space versions, which might slow it down the storing of any alterations.

8) Encoding text

CODE

```
private byte[] encode_text(byte[] image, byte[] addition, int offset)
{
 if(addition.length + offset > image.length)
{
 throw new IllegalArgumentException("File not long enough!");
}
for(int i=0; i<addition.length; ++i)
{
int add = addition[i];
for(int bit=7; bit>=0; --bit, ++offset)
{
 int b = (add >>> bit) & 1;
  image[offset] = (byte)((image[offset] & 0xFE) | b );
}
}
   }
```

A byte's bits are ranked from most significant to least significant, with the left most bit being the most significant and the right most bit being the least significant. This gives us the key. if we need to alter any data in this picture, we want it to be as inconspicuous, if not invisible, as possible. As a consequence, we want to change the least significant bit of some of the pixels. We alter each byte in this manner, with a highest value of 1.

This code does this in the following way:

To loops over the addition array's bytes

```
for (int i=0; i<addition.length; ++i);
```

To assign add to the current byte

```
int add = addition[i];
```

To iterates through the eight bits of the byte held in add

```
for (int bit=7; bit>=0; --bit, ++offset);
```

The value of the byte add shifted right bit positions AND 1 is assigned to b.

```
int b = (add >>> bit) & 1;
```

This may sound complicated, but the end result is a loop in which b is assigned the next single bit value of the byte add, either 0, or 1.

This is best demonstrated by the following illustrations:

We will start with int b = (add >>> bit); only, Say:
add = 87 = 01010111
First loop through, bit = 7:
01010111 = 87
>>> 7
00000000 = 0
Next time, bit = 6:
01010111 = 87
>>> 6
00000001 = 1
Next time, bit = 5:
01010111 = 87
>>> 5
00000010 = 2
Next time, bit = 4:
01010111 = 87
>>> 4
00000101 = 5
… and so on.

You can how the right bits match the left bits of add, in a growing number based on how many positions we shift include.
Now to apply the & 1:
First loop:
00000000 = 0
00000001 = 1
00000000 = 0 = b
Next:

00000001 = 1
00000001 = 1
00000001 = 1 = b

Next:
00000010 = 2
00000001 = 1
00000000 = 0 = b

Next:
00000101 = 5
00000001 = 1
00000001 = 1 = b

Depending on the last bit of the shifted add byte, b is assigned a value of 0 or 1. We achieve the same result as previously by performing AND by one, which helps clear all bits to zero save the last, which remains untouched. This means that the value of b represents the bit at location bit in the for loop.

image[offset] = (byte)((image[offset] & 0xFE) | b);

This piece of code works in a similar way. 0xFE is a hex value that correlates to the binary number 11111110. The first 7 bits will remain untouched, but the least critical bit will be set to 0. Then, with the last bit 0, we OR it with b, which might be 00000000 or 00000001. The last bit will be set to the value contained in b as a result of this. Because the OR operation with 0s does not change any of the first 7 bits, and thus knowing that the last bit may be a 0, the value in this location of b is sure to be placed here, whether this is or 1.

As the loop runs, the code moves the offset value, distributing the eight bits of a byte of addition among 8 least significant bits of 8 different and following bytes of the image. It is also crucial that we encode the length first and in a static way, for example, saving it in 4 bytes or the first 32 least significant bits. As a consequence, we know how many least significant bits to read after the length in order to obtain the complete message.

9) DecodeText:

CODE

```
private byte[] decode_text(byte[] image)
{
  int length = 0;
int offset = 32;
  for(int i=0; i<32; ++i)
 {
```

```
 length = (length << 1) | (image[i] & 1);
 }
byte[] result = new byte[length];
for(int b=0; b<result.length; ++b )
{
for(int i=0; i<8; ++i, ++offset)
    {
 result[b] = (byte)((result[b] << 1) | (image[offset] & 1));
 }
 }
return result;
}
```

The process may look straightforward at first glance, but I'll explain how each step works to obtain the bits we encoded.

- **int offset = 32;** – Because the message length is recorded as a 4 byte integer, or 32 bits, the message starts after 32 bytes of image.

- for(int i=0; i32; I - Because the first 32 bytes contain one bit of our length, we must loop over all 32 bytes to get the length.

- **length = (length 1) | (image[i] & 1) -** We shift the length bits to the left by one, then OR the result with the least significant bit of the image byte. (& 1) clears all bits except the final one, which remains untouched. Bits are moved forward and placed in the newly vacant least significant length slot as they are added**.**
  For the same reason as the bit conversion array being {0, 0, 0, byte0}, This for loop may be written with i=24 and still work. Both of these elements were kept out of the final code since having the bigger ranges allows for more text to be buried in the image.

- **for(int b=0; bresult.length; ++b) –** Now that we have a length and a byte array to hold the data, we loop over that many picture bytes.

- **for(int i=0; i8; I ++offset) –** We must loop through the 8 bits of a byte one more time to gather them.

- **result[b] = (byte)((result[b] 1) | (image[offset] & 1) -** The resulting byte array is made up of the least significant bit from each subsequent byte. Now that the loops are properly constructed, this is acquired in the same way as the length.

That explains the enigma of Steganography. Of course, there are various methods, and most typically, the text is encrypted before it is buried to reduce the likelihood of it being noticed and/or cracked. The more random the image, the easier it is to add data to the image without being detected.

```
public String encrypt(String value,String key)
  throws Exception {

        byte[] raw = key.getBytes(Charset.forName("US-ASCII"));
        if (raw.length != 16) {
          throw new IllegalArgumentException("Invalid key size.");
        }

        SecretKeySpec skeySpec = new SecretKeySpec(raw, "AES");
        Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
        cipher.init(Cipher.ENCRYPT_MODE, skeySpec,
            new IvParameterSpec(new byte[16]));
        byte [] encrypted_data=cipher.doFinal(value.getBytes(Charset.forName("US-ASCII")));
        BASE64Encoder b = new BASE64Encoder();
       String  encryptedString = b.encode(encrypted_data);
        return encryptedString;
}
```

Above were the example of importing coding concept used to develop the proposed system. Below I will share the actually code snippets.

AES Encryption algorithm

```
public String decrypt(String encrypted_data,String key)
    throws Exception {
          System.out.println("The AES Key is :"+key);
          BASE64Decoder b = new BASE64Decoder();
          byte[] encrypted=b.decodeBuffer(encrypted_data);


          byte[] raw = key.getBytes(Charset.forName("US-ASCII"));
          if (raw.length != 16) {
            throw new IllegalArgumentException("Invalid key size.");
          }
          SecretKeySpec skeySpec = new SecretKeySpec(raw, "AES");

          Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
          cipher.init(Cipher.DECRYPT_MODE, skeySpec,
              new IvParameterSpec(new byte[16]));
          byte[] original = cipher.doFinal(encrypted);

          return new String(original, Charset.forName("US-ASCII"));
    }
```

1 References

[1] "Jcreator," 1 11 2021. [Online]. Available: http://www.jcreator.org/download.htm.

[2] "java bascis," 1 11 2021. [Online]. Available: https://www.javatpoint.com/java-basics.

[3] "what is AES," 1 11 2021. [Online]. Available: https://www.cloudwards.net/what-is-aes/.

[4] "LSB based image steganography," 1 11 2021. [Online]. Available: https://www.geeksforgeeks.org/lsb-based-image-steganography-using-matlab/.