

COMP 7615 Final Report  
Stega-Yes-Graphy  
Thilina Ratnayake  
Justin Tom

# Table Of Contents

[Assignment Guidelines](#)

[Objective](#)

[Mission](#)

[Steganography](#)

[How is Steganography done?](#)

[Least Significant Bit \(LSB\) Method](#)

[Implementation](#)

[Back End Server](#)

[Front End](#)

[Encryption](#)

[Pseudo Code](#)

[Decryption](#)

[Pseudo Code](#)

[State Machine Diagram](#)

[Tests and Analyze](#)

[Conclusion](#)

[Limitations](#)

[Future Implementations](#)

# Assignment Guidelines

## Objective

To create a web-app which would allow users to hide and extract messages within images using Steganography. Also, to use encryption to further protect the information.

## Mission

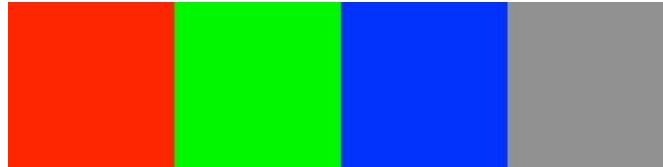
Study and utilize the tools and practices learnt from class in order to work on a project revolving around the idea of steganography web application. Use it as a means for “hidden” communication between two users. The web application will allow a user to enter in their text and generate an image to use in “regular” communications. Take user’s data (text) and convert it into bits and employ the use of cryptography to encrypt the data bit stream to further hide the data. Using the LSB steganography method to hide the data in an image. Send the steganography image to the other user. Receiving user uses the same technique of LSB to uncover the data (text). This will all be done within a web application environment to allow for a more convenient use of either computer, or mobile devices such as tablets and smartphones.

## Steganography

Steganography come from the Greek words “Covered writing” - it is the art of writing in cipher. It is both the art and science of hiding information such as text, images, PDF, etc. within a larger data medium (images, sounds, videos, text, etc.). It hides the information in plain sight - such as if one were to hide a text message in an image, it would simply show an image. Therefore, this also means that people who are not the intended recipients won’t notice it, since most other users who see it would just assume that it is simply another image without giving it a second thought. Some possible uses for steganography include digital watermarking (copyright protection) in order to help prove the material belongs to you if other users try to steal your photos. Hiding sensitive data if you want to send some information over to someone but don’t want to have it shown in plain text in the case that someone else sees or intercepts it, you can hide it within a medium such as an image. Sometimes even when encryption is not permitted or possible, steganography can be used as an alternative method to hide the information. Often however, steganography is also used to complement/supplement encryption, so even if the encrypted file is discovered, the hidden message is not seen or vice versa where the stego data is discovered, it still cannot be viewed as it is encrypted.

# How is Steganography done?

Steganography can be done through various different methods, however, for this project, we employed the most basic method - least significant bit (LSB) method. Images are composed of hundreds of thousands to possible millions of pixels. Each color pixel is composed of three basic (sometimes four, alpha channel) color values – red, green and blue (RGB) and are represented with one byte each.



These values range from 0 – 255 in decimal value and '00000000' to '11111111' in binary. The idea is to take the last, least significant bit in the binary value and change it to either a 1 or 0 depending on the data bit we are replacing it with. Through this method, the worst case scenario for a RGB value to change by is 1 ( $0 \rightarrow 1$  or  $1 \rightarrow 0$ ), which is not visible or detected through a human eye. However, the more data that is stored/hidden in the image, the more the image will be altered and differ from the original and consequently more noticeable and detectable. A way to prevent this would be to use an image with lots of noise on it already – however this is quite suspicious to use.

## Least Significant Bit (LSB) Method

For example we have 3 pixels:

Pixel 1: (214, 23, 28).

Pixel 2: (228, 26, 48).

Pixel 3: (207, 17, 19).

	Pixel 1	Pixel 2	Pixel 3
Red	11010110	11100100	11001111
Green	00010111	00011010	00010001
Blue	00011100	00110000	00010011

Now say we want to conceal the following set of 9 bits: 101101100

	Pixel 1	Pixel 2	Pixel 3
Red	11010111	11100101	11001111
Green	00010110	00011010	00010000
Blue	00011101	00110001	00010010

As you can see, we place each bit in the 9 bit string sequence into the least significant bit position in the color's binary value. For the first three bits, 101, each bit in the least significant bit position for the first pixel is changed, as the original value was 010. For the second set of 3 bits, 101, both red and blue's LSB were changed but the green pixel's LSB stayed the same as its original value was 000. The same principles apply for the last pixel, where the new values 100 only change the green and blue LSB values since the original LSB values were 111.

## Implementation

The programming languages used to power the web application were HTML, CSS, Javascript, and NodeJS. As for encryption, we used the leading standard of AES-256 encryption in order to further secure the data before hiding it in the image, so as to if an attacker were to somehow realise the image was steganography and was able to extract the data from the image, they would still not be able to read the data as it would be encrypted. The web application uses a front end website which connects and communicates back and forth between a back end server.

## Back End Server

The back end server was powered by NodeJS and Express. The server was in charge of setting up a virtual private server (VPS), creating API endpoints for the front end to send and receive data to and from, create the main functions to employ the encryption and decryption algorithm for the steganography and any other helper functions.

## Front End

The front end was powered by HTML, CSS and JavaScript. This front end was in charge of creating the main interface view (index.ejs), creating JavaScript to handle page animations as well as JavaScript to handle file upload & posts to the back end server.

## Encryption

The encoder algorithm will take in three variables when executing the method. The arguments are the cover image (bitmap format), the data the user wishes to hide (text message), and the

encryption key the user wishes to use in coordination with the AES encryption. It will also check to make sure that the data you are trying to put into the cover image is small enough to be hidden inside (the logic of taking the cover image's dimensions and multiplying the width and height to get the total number of pixels and then multiplying that by 3 - as the program stores 3 bits per pixel (RGB) and adding the size of the crafted "header" and then subtracting the total size of the message from that). The output file will also be in a bitmap format. It will also hide the filename, extension and file size of the data in an artificial "header" delimited by a null character terminator. This will allow the decoder program to read and take in the filename as well as the size of the data in order to properly read the exact data hidden in the image instead of reading unrelated bits to the hidden data. The program essentially takes the specified text and encrypts it with the specified AES encryption key and then converts it from ASCII into a lengthy string of binary bits which is then has each bit swapped with the LSB of the RGB values as it iterates through each pixel in the image until there is no more data left to hide. It will then output the new stego image to the user for them to save and send to their intended recipient.

## Pseudo Code

```
35 4. To perform steganography
36   #1. First create the header
37   header = createHeader("text",message.length)
38   #2. Encrypt the command
39   message = encrypt(message)
40   #3. Convert message to ascii
41   ASCIIarray = messageToBits(message)
42   #4. Convert ASCII values (integers) into 8-bit binary
43   embedText = toBinary(ASCIIarray)
44  #5. Do the same for the header
45   ASCIIarray = messageToBits(header)
46   header = toBinary(ASCIIarray)
47   #5 Combine header + text
48   embedData = header + text
49  #6. For every pixel in image, iterate through it
50  Image.scan(x,y){
51      #Grab the red, green and blue pixel.
52      red = pixel(x,y)
53      var redValue = red
54      #Change the last bit of the redvalue to be what is in the array
55      var redValue[7] = embedData.push();
56      red = redValue
57      ##Repeat for Green & Blue
58  }
```

## Decryption

The decoder program will take three variables when executing the method. The arguments are the steganography image (the one with the hidden data inside), and the decryption key (which needs to match exactly the encryption key in order to properly decrypt the message). This algorithm will open the image and iterate through each pixel and its RGB values, converting it to binary and grabbing only the last bit (the only bit which matters) and concatenates it into a long string. After it goes through and extracts the header with the filename and file size in it, using delimiter of null character, it will continue on the real data of the hidden file. After it has the long binary string with the data of the hidden text, it is then converted into ASCII and then decrypted using the specified decryption key and outputs the hidden text in plaintext to the user.

## Pseudo Code

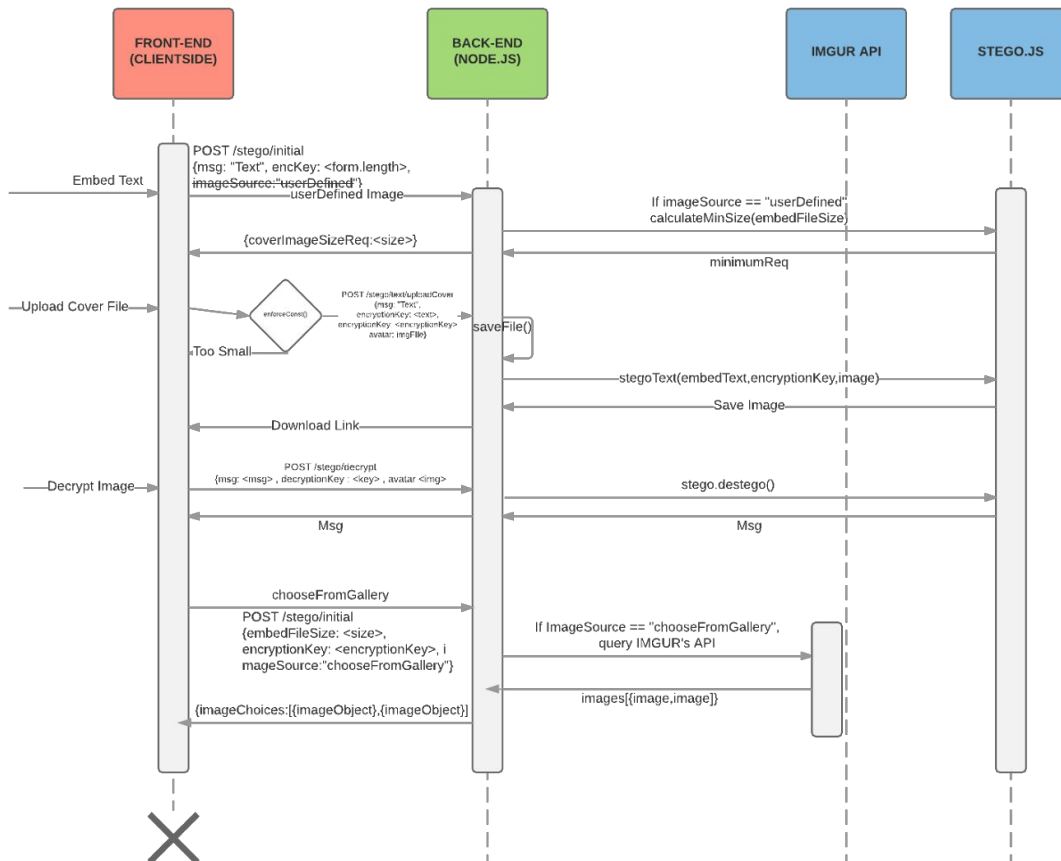
```
69 5. Destego
70 ✓ destego(coverImage, decrpytionKey){
71     #1. First scan the image and get the header
72     array bitArray
73 ✓     image.scan(x,y){
74 ✓         #Grab last bit of RED, GREEN, BLUE pixel
75         bitArray.push(pixel(x,y).red[7])
76         bitArray.push(pixel(x,y).green[7])
77         bitArray.push(pixel(x,y).blue[7])
78     }
79     #2. Split into 8 bit chunks
80     array chunks = bitArray.split(8)
81     #3. Convert the first 100 chunks into integers
82     array integers = toIntegers(chunks)
83 ✓     #4. Convert integers to ascii
84     array ascii = toASCII(integers)
85     data = ascii.join("")
86     #5. Scan for the header, it is delimited by \n's
87     elements = data.split(\n)
88 ✓     #6. Parse array elements
89     type = elements[0]
90     length = elements[1]
91 ✓     #7 Once we know the length, go through the image again and get the full data
92     startingPoint = ("text" + \n + "length" + \n + 1)*8
93     endingPoint = startingPoint + (length)*8
94     counter = startingPoint
95     array dataArray
96 ✓     image.scan(x,y){
97 ✓         if(x+y >= startingPoint){
98             #Grab the last bit of the RGB values and add it into array
99             array.push(red[7])
100             array.push(green[7])
101             array.push(blue[7])
102         }
103     }
104     #8 Chunk it
105     array chunks = array.split(8)
106     #9 Convert to Integers (same as above)
107     #10 Convert to ascii (same as above)
108     #11 Send to user
109 }
```



# State Machine Diagram



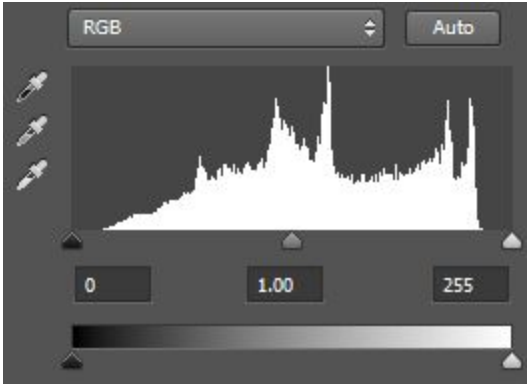
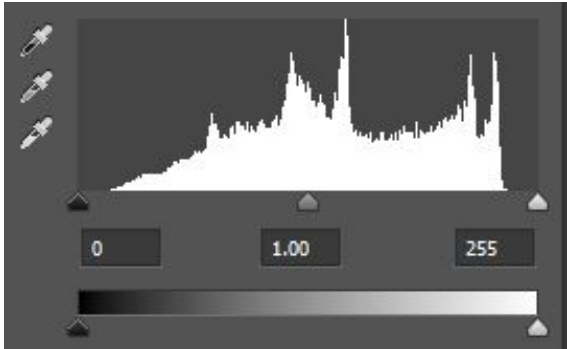

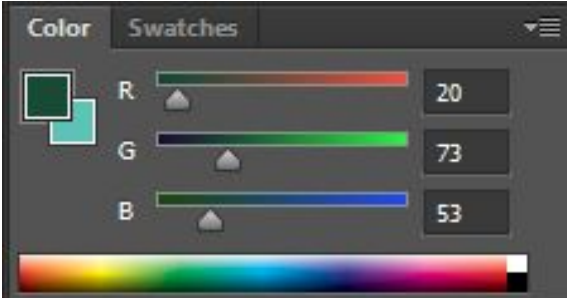
## STEGO-YES-GRAPHY SEQUENCE DIAGRAM

Thilina Ratnayake & Justin Tom | Rev: 22 Nov 2015



## Tests and Analyze

	Original Image	Stego Image
--	----------------	-------------

Image		
Histogram		
First Pixel's RGB Value		

Test	Time Taken	Figures
Encrypting an image with 50 bytes of data into an image.	2.23s (Time taken from button click to displaying the stego image)	1.1, 1.2
Encrypting an image with 10,000 bytes of data into an image.	2.30s (Time taken from button click to displaying the stego image)	2.1, 2.2
Decrypt an image with 50 bytes of data from a stego'd image.	2.02s (Time taken from button click to displaying the original image)	3.1, 3.2

	displaying the hidden message)	
Decrypt an image with 10,000 bytes of data from a stego'd image.	2.30s (Note: the string was too large to output to the user so the time was the time taken from button click to server response)	4.1
Encryption of an image with encryption key of 250 bytes and 250 bytes of data.	2.08s (Time taken from button click to displaying the stego image)	5.1, 5.2
Decryption of a stego image with a decryption key of 250 bytes and 250 bytes of data.	1.99s (Note: the string was too large to output to the user so the time was the time taken from button click to server response)	6.1



All tests were done using the same CPU, internet connection, encryption/decryption key and the above cover image to ensure high internal validity for each test.

### Encryption of Message

**Message:**

**Encryption Key:**

**Select image to stego:**  Bliss.bmp

Figure 1.1 - The page displaying the 50 bytes of data for the first test and the bliss bitmap image.



Figure 1.2 - The stego'd image after encryption of the 50 bytes of data



## Encryption of Message

**Message:**

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam fringilla tempor rutrum. Suspendisse luctus varius augue eget varius. Donec vel hendrerit sapien, sed bibendum lacus. Fusce pretium euismod ex a aliquet. Vivamus dictum, urna vitae ullamcorper placerat, dui est ultrices nisl, id molestie augue mi vitae mauris. Etiam id nisi et odio ornare tincidunt. Etiam venenatis pulvinar tortor. Suspendisse ut neque mattis, vehicula felis semper, dictum augue. Vivamus efficitur, dolor non pharetra fringilla, lectus risus condimentum lacus, aliquet interdum sem dui eget leo. Mauris rhoncus aliquam ultrices. Vivamus congue condimentum purus at tristique.

Pellentesque finibus quam lacus, quis vehicula mi pretium non. Sed tempus ac mauris et interdum. Nulla facilisi. Pellentesque accumsan rutrum lacinia. Donec bibendum faucibus felis, ut pellentesque tortor vehicula sed. Fusce id sem neque. Maecenas semper finibus felis, quis dictum sem cursus porta. Aenean scelerisque quis est et vulputate. Nulla dapibus, risus et dignissim laoreet, quam metus interdum justo, vel dapibus turpis nisi in massa.

Cras eget condimentum mi. Mauris maximus tristique nulla vitae viverra. Sed gravida nibh risus, sit amet imperdiet elit ultricies nec. Quisque ornare nisl sit amet augue condimentum, vel malesuada ipsum maximus. Nullam urna velit, pretium at leo in, lacinia egestas augue. Donec molestie justo vitae nunc sodales volutpat. Praesent ut risus ut neque sollicitudin maximus. Cras eu posuere ligula. Nam

**Encryption Key:**

....

**Select image to stego:**

Bliss.bmp

Figure 2.1 - The page displaying the 10,000 bytes of data for the first test and the bliss bitmap image.



Figure 2.2 - The stego'd image after encryption of the 10,000 bytes of data

### Decryption of Message

Decryption Key:

Selected stego image:  6866f614dc80d...17f91df52.bmp

Figure 3.1 - The page displaying the 50 bytes of data stego image for the third test and the decryption key.

### Decryption of Message

Decryption Key:

Selected stego image:  6866f614dc80d...17f91df52.bmp

Decrypted Message: Lorem ipsum dolor sit amet, consectetur cras amet.

Figure 3.2 - The page displaying the outputted 50 byte hidden message from the stego image.

### Decryption of Message

Decryption Key:

Selected stego image:  afac8565c20cb...538dd9363.bmp

Figure 4.1 - The page displaying the 10k bytes of data stego image for the fourth test and the decryption key.

### Encryption of Message

Message:

Encryption Key:

Select image to stego:  Bliss.bmp

Figure 5.1 - Encrypting the 250 bytes of data with 250 byte encryption key.



Figure 5.2 - The stego image result of the 250 byte encryption key.

The image shows a web application interface with a dark background. At the top, the title "Decryption of Message" is displayed in white. Below the title, there are two input fields. The first is labeled "Decryption Key:" and contains a long string of asterisks. The second is labeled "Selected stego image:" and contains a file path "c2fd4688e681b...77d8ee99.bmp" next to a "Choose File" button. Below these fields is a white button with the text "DECRYPT IMAGE" in black.

Figure 6.1 - The page displaying the hidden message with a 250 byte decryption key.

For the tests, we also found that good examples included images that were very colourful and had lots of different shading and lighting in the photo or less useful, images that already had lots of “noise” within the photo. This would help disguise the steganography data within the image better, as the changes wouldn’t be as noticeable. Bad examples/images to use as the cover image would include any solid color pictures. Though the method we use only alters the least significant bit, it would still make a slight difference when comparing the original and stego images side by side - however this is usually never the case.

From analyzing the multiple tests we ran against the web application, we didn’t notice much of a performance or timing issue when attempting to encrypt more data. This would be a result of the

fact that our web application currently only takes in text as data rather than other larger mediums such as images or videos. If that were the case, then the performance might be slightly reduced - however, since we only deal with text the performance and time difference is minimal. The same results were found when using a larger encryption/decryption key for the data. Though the tests were not as extensive as they could have been, the preliminary results showed no real difference from increasing/decreasing the size of the encryption keys.

## Conclusion

The web application is fairly basic and was originally intended for a proof of concept program and is in no way perfected. Although this web application is very basic, principally through the use of the least significant bit method of steganography, and therefore not the best or most discrete it can be, it still carries a lot of potential of expansion of features and usage - especially since the web application can be accessed and used on mobile devices (on top of web browsers on PCs).

## Limitations

The limitations of our web application are that the images used as the cover photo must be of bitmap format in order for the application to properly function. Also, due to the webpage being a state, it only allows the user to encrypt and decrypt once before requiring a page refresh if you would like to stego/de-stego more images. Also as the current program rests, it only allows for text data to be hidden inside other mediums.

## Future Implementations

Future implementations would be to change it so the LSB isn't changed for every pixel in succession of each other, but instead implement a more complicated algorithm to insert the data such as every third pixel or even putting it diagonally in rather than on the X or Y axis. Another future implementation would be possible use of the alpha channel (opacity/transparency) would have also been something to take into factor, as it would have made it easier to encode and decode as each byte would go into two pixels evenly since there would be four bits per pixel (RGBA) instead of the current three (RGB). This would also allow for larger data to be hidden inside the image medium (one more bit per pixel).

Other future work that we would have like to have completed would be to incorporate support for multiple different image format as the cover image, rather than just bitmap format images to allow for more flexibility, universality and freedom for the user. Along the same lines, we would have like to allow for the functionality of using other data types to hide (rather than just text) such as other images, videos, PDFs, etc. as well as new and other mediums to have the data hidden in.



We were also unable to fully add the feature of incorporating the Imgur API due to time constraints. However if we had been able to, the functionality would allow the user to choose an image from a randomly generated selection of them from the image hosting site Imgur, instead of having to upload and choose their own bitmap images.