Steganography – Embed Text Message Inside An Image

Suyog Siddheshwar Swami (1001119101)

May 10, 2015

1. Introduction:

Steganography is an art and science of writing hidden messages in such a way that no one apart from the intended recipient knows of the existence of the message.[1] It is an adaptation of a combination of two Greek words *Steganos*: which means 'concealed', 'coveted' or 'protected'. And *graphien*: which means 'writing' [2]. Hence steganography is an art of hidden writing of a message. The message can be hidden inside any file such as a text, image, audio or video. Also the message to be hidden can be any file like text, image, audio or video. Steganography is often used in situations where the actual existence of the message needs to be obscured [1]. The advantage of steganography over cryptography is that the intended secret message does not attract attention to itself as an object of scrutiny [2]. If a plain text is encrypted even though it can be unbreakable, it attracts the attacker or any unwanted person. This is where stenography plays a crucial role in hiding the existence of the message.

As stated above steganography is used in situations where existence of the image is to be kept secret. Whereas cryptography is used in situation where existence of message is known but the meaning of the message is made obscured.

First examples of steganography dates as back as during Ancient Greece, where Histiaeus[4], a prisoner of a rival king, needed to communicate a secret message to his own army. In a way to do this he shaved the head of a willing slave and tattooed his message on the scalp. Once the slave's hair had grown back, he was sent to deliver the hidden writing in person.

2. Types of Steganography:

Steganography in digital world can be implemented anywhere and everywhere where bits are involved. In digital steganography the goal of hiding the data is achieved by embedding the bits of that data inside the bits of the source media such that the existence of the hidden data is not detected.

Following are some types of the steganography in digital world:

a. **Audio Steganography**:

The data can be embedded by creating an echo (Echo Data Hiding)[5] or at a different frequency which is not heard by human ear (Phase Coding)[5].

b. **Image Steganography**:

The LSBs [1] of RGB pixel values of source image are changed by the bits of the intended message.

c. Video Steganography[6]:

This can be viewed as a combination of audio and image steganography. We can embed message in each frame of the video or in the audio of the video file. Video Steganography gives a large area for hiding data and thus detection becomes extremely difficult. Steganosauraus[6] is dissertation project of James Ridgway which focuses on video steganography.

d. **DNA Steganography[7]**:

This stenographic type is used in secured printing in which the publication needs to authenticate its printed material. An ink with specific DNA pattern is used while printing. A pirated book is detected by checking the DNA of the suspicious book with the DNA of authenticated publisher's ink.

3. RGB Color Model[3]:

RGB stands for Red-Green-Blue. RGB color model is used to display images in electronic system. Basically each pixel of an image is a combination of 3 values of Red Green and Blue respectively. A value (0,0,0) represents black color, (255,255,255) represents white color, (255,0,0) represent Red color, (0,255,0) represent Green color and (0,0,255) represent Blue color. Any values in between with different combinations represent different colors.

In today's modern images there are there are more coordinates added to the pixel value to produce different effects in the image. HSL is one such extension of RGB model. HSL[8] stands for Hue-Saturation-Luminance. Also a fourth coordinate "alpha" is added in some images to provide opacity for the image. This model is RGBA where A stands for Alpha.

4. Steganography using Image as media:

When Image is used as a media for embedding text, image or any other media or data file the RGB model plays an important role in this. The basic algorithm for Image Steganography goes as follows [1]:

A. Convert the text or the file to be embedded into its binary form
B. Group the binary values into a group of three n bit values
C. Iterate through the image pixel by pixel
D. Convert the RGB value of a pixel where want to embed the data to its binary value.
E. Now change the n LSBs of the RGB values with the n bit values of each grouped binary in step B.
F. Repeat above steps till the all data is embedded

5. Security:

But if the above algorithm is used for embedding the data we are embedding it raw. We are not providing any security to the data. Even though we are hiding the data inside the image, but if by any means the attacker is successful in extracting the data the data is not secure. Hence, in order to incorporate security along with steganography we use encryption to encrypt the data in this case text and then embed the encrypted image [1]. Hence the security for the embedded message is provided by the encryption algorithm we use. For our project we are using RSA encryption algorithm. As we know RSA is an asymmetric key encryption algorithm, it uses a public-private key pair. It can be beneficial in using this algorithm as we will embed message using receiver's public key and send it to receiver where he will need to decrypt the message using his private key.

6. The Algorithm:

Following is the algorithm which I have used while implementing this project. It is somewhat different that the one I showed during the presentation. The issues which made me change the algorithm are stated in below sections.

Embedding:

A. Encrypt the text message to be embedded using RSA encryption
B. Convert the list obtained as cipher text into a string of integers
C. Store the cipher text list length in one variable
D. Store the Cipher text integer element length in a list
E. Convert each integer(0-9) in the string into its 6 bit binary
F. Group the 6 bit binary values into 3 groups of 2 bits each
G. Store these 3groups output as a list in list of all the integers
H. Now start from a predetermined pixel point on an image and start iterating
I. Convert the RGB pixel value of the pixel to its 8 bit binary
J. Replace value of 2LSB of R with first group of 1^{st} integer binary value.
K. Replace value of 2LSB of G with first group of 2^{st} integer binary value.
L. Replace value of 2LSB of B with first group of 3^{st} integer binary value.
M. Repeat steps F-J till the cipher text string is over
N. Similarly store cipher text length obtained in step C. at top right corner of the image
O. Similarly store the Cipher text integer element list at the bottom left of the image.

Extracting:

A. Go to the top left of the image and get the RGB value of the pixel
B. Concatenate 2 LSBs of the RGB to form an integer which is the length of the cipher text list
C. Now go to bottom left and start iterating, get the RGB pixel value
D. Concatenate 2 LSBs of the RGB to form an integer which is the length of the 1^{st} cipher text element.
E. Append the integer values to a list and repeat C-D till the value is equal to that obtained in B.
F. Start iterating from the predetermined point(center) of the image.
G. Get the RGB value if at that pixel
H. Convert each RGB to its 8 bit binary
I. Get the 2 LSB if each RGB
J. Concatenate the 2 LSBs of each RGB and form an integer
K. Append this integer to a list
L. Repeat steps A-F till the value equal to sum of values obtained at E.
M. In the end form a list using integer list at K such that it is similar to the original cipher text.

7. Issues Faced while implementing:

Following discussed are the major issues faced while implementing the project which led to modifying the algorithm

i. Previously I had planned to randomize the placement of embedding in the image by skipping the number of pixels based on the value we store at the B of previous pixel. But after the presentation when I discussed this with Brian who is also doing similar project, I decided to remove the skipping as the changes in the bits were negligible. Also randomizing puts an unnecessary load of keeping track of the previous pixel's B value.

ii. Also, during the presentation I mentioned that I will take 2 integers and compare them with 64(as maximum 6 bit value is 64) to make them fit in 3 groups of 2bits. But if the value was greater than 64 I was taking only 1 bit. Hence if the integers in cipher text were as follows:

….2345632908….

I used to get output as

….234563298….

The zero used to get removed while binary to integer and vice versa conversion. This was a very bad issue as it changed the cipher text and gave completely absurd results. Hence I planned to take each digit and convert it into 6 bit binary. Hence now the largest value will be 9.i.e 001001, the 2LSB added to R are always 00.

8. Modifications tried:

i. As you can see in the code I have stored the lengths of the cipher text elements in the image as 3 bit versions as I pointed out during the presentation. Hence I have used 2 LSBs and 3LSBs in this project. As we can see in the images output there isn't any visible difference when 3LSBs are used, because 3LSBS change a color proportion by maximum of 7. If the image is rich in pixels the use of 3 LSBs isn't visible

ii. Also I have tested all the inputs and outputs on different length of messages. The image can cover a count of $1/4^{th}$ of the pixel count as I and iterating from center to the bottom right. This can be changed to accept more text by changing iteration from.

iii. I have tested the outcomes on PNG and JPG image formats. Both are lossless image compression formats hence the data written in the image is not lost during the compression [1].

9. The code:

The code is attached along with the report.

Requirements:

- The main property to remember while choosing the image as an input to our program is it deals with only RGB values. Hence if any image has RGBA model an error will occur while running. Hence we should use images which are not having opacity in them. Some sample images to test are attached along with.
- Python 3.4 should be installed.
- Pillow 2.8.1 is a Python Image Library which is required.
- Numpy 1.9.0 is required.

If these 2 libraries are present we can successfully run the program

Some useful snippets are as follows (They are self-explanatory with comments alongside):

For loading Image Pixel data into an array.

```
im=Image.open("tajmahal.jpg")      #"6.png"                    #Image used for embedding text message
im.show()
r, g, b = np.array(im).T                                       #Image pixel data stored in numpy under 3 coordinates i.e. RGB
```

For inserting text data into the pixel RGB values

```
for i in range(int(im.size[0]/2),im.size[0]-1):                #Inserting ciphertext inside the image
    for j in range(int(im.size[1]/2),im.size[1]-1):
        if m <len(small_str_list):
            bitss=textwrap.wrap(small_str_list[m],2)           #textwrap is used to group the six bit binary into 3 groups of 2 bits each

            rr=textwrap.wrap(str(format(int(r[i][j]),'#010b')[2:]),2)   #Converting the pixel R value to 8 bit binary
            rr[3]=bitss[0]                                     #changing the last bit is R with first 2 LSBs of messages binary number.
            r[i][j]=int(''.join(rr),2)                         #converting binary to integer which will form R value of new image

            gg=textwrap.wrap(str(format(int(g[i][j]),'#010b')[2:]),2)   #Same procedure as above for G value of the pixel
            gg[3]=bitss[1]
            g[i][j]=int(''.join(gg),2)

            bb=textwrap.wrap(str(format(int(b[i][j]),'#010b')[2:]),2)   #Same procedure as above for B value of the pixel
            bb[3]=bitss[2]
            b[i][j]=int(''.join(bb),2)

            m=m+1
        else:
            break
```

Extracting data from RGB pixel values

```
for i in range(int(img.size[0]/2),img.size[0]-1):                        #go to the center of the image and start extracting the RGB values
    for j in range(int(img.size[1]/2),img.size[1]-1):
        if t<f_total:
            final_r_bin_w=textwrap.wrap(format(int(r1[i][j]),'#010b')[2:],2)
            final_g_bin_w=textwrap.wrap(format(int(g1[i][j]),'#010b')[2:],2)
            final_b_bin_w=textwrap.wrap(format(int(b1[i][j]),'#010b')[2:],2)
            final=int(str.join('', (final_r_bin_w[3],final_g_bin_w[3],final_b_bin_w[3])),2) #Unwrap the 3LSBs to get the cipher text value
            final_str.append(final)
            t=t+1
        else:
            break
```

10. Snapshots:

Message and Encrypted cipher text to be embedded in the image

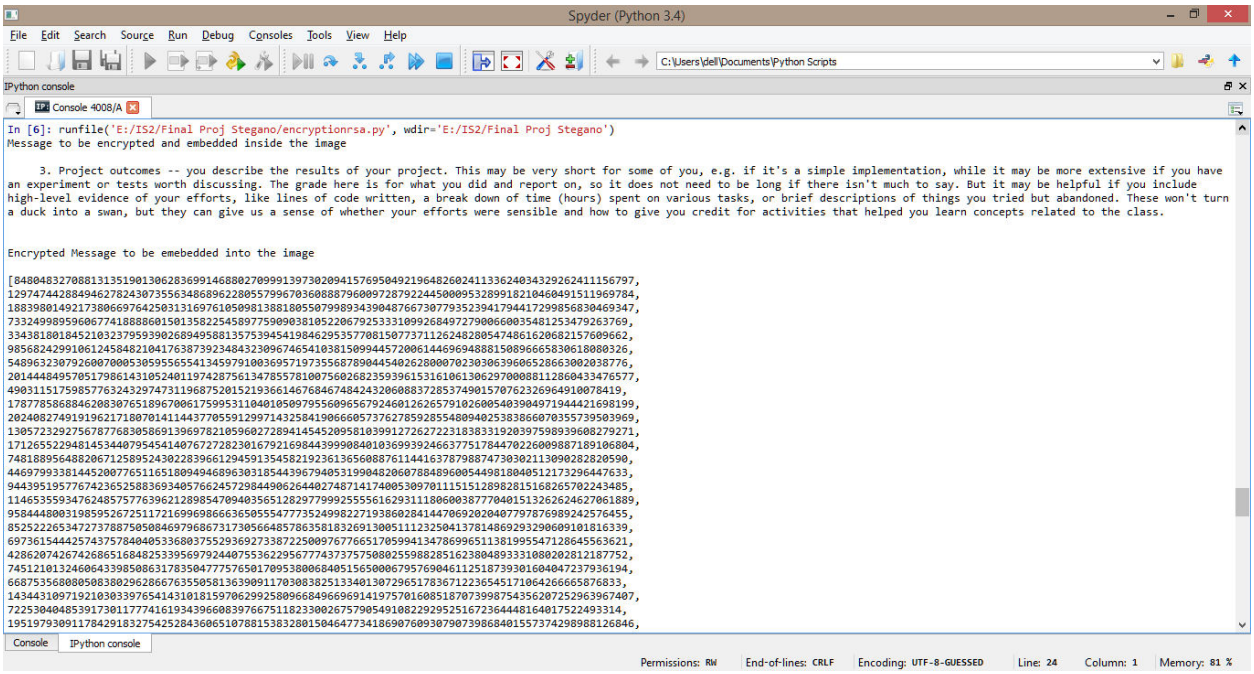
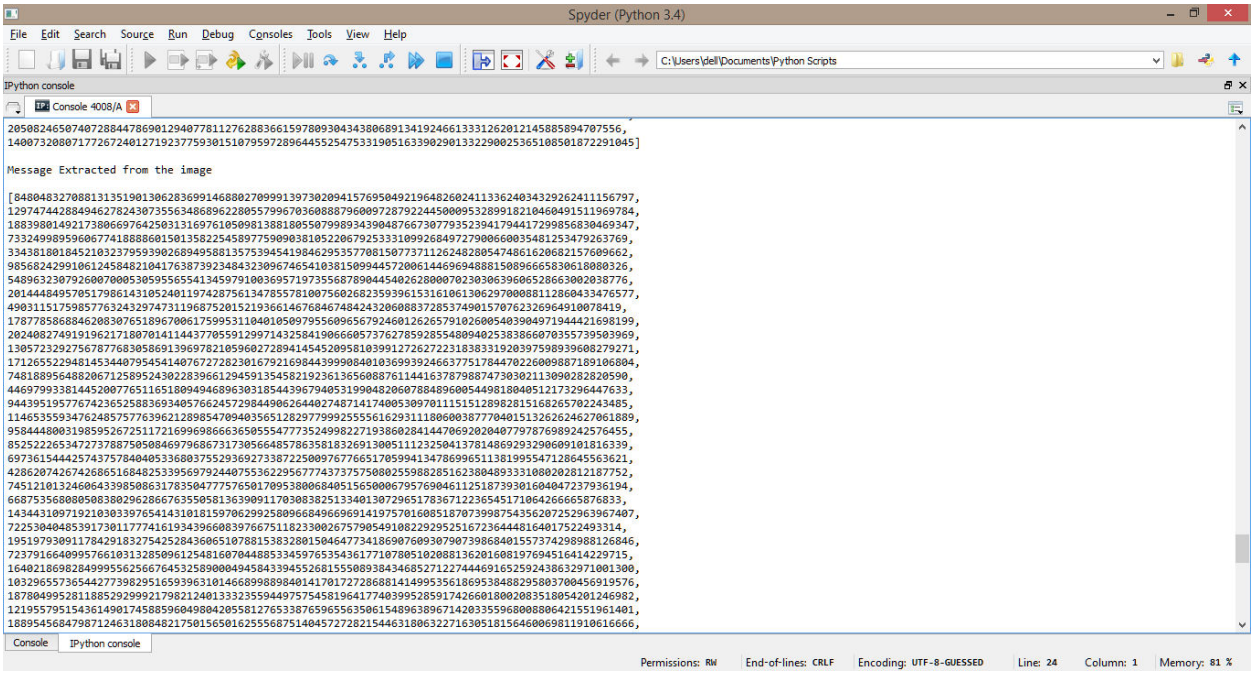


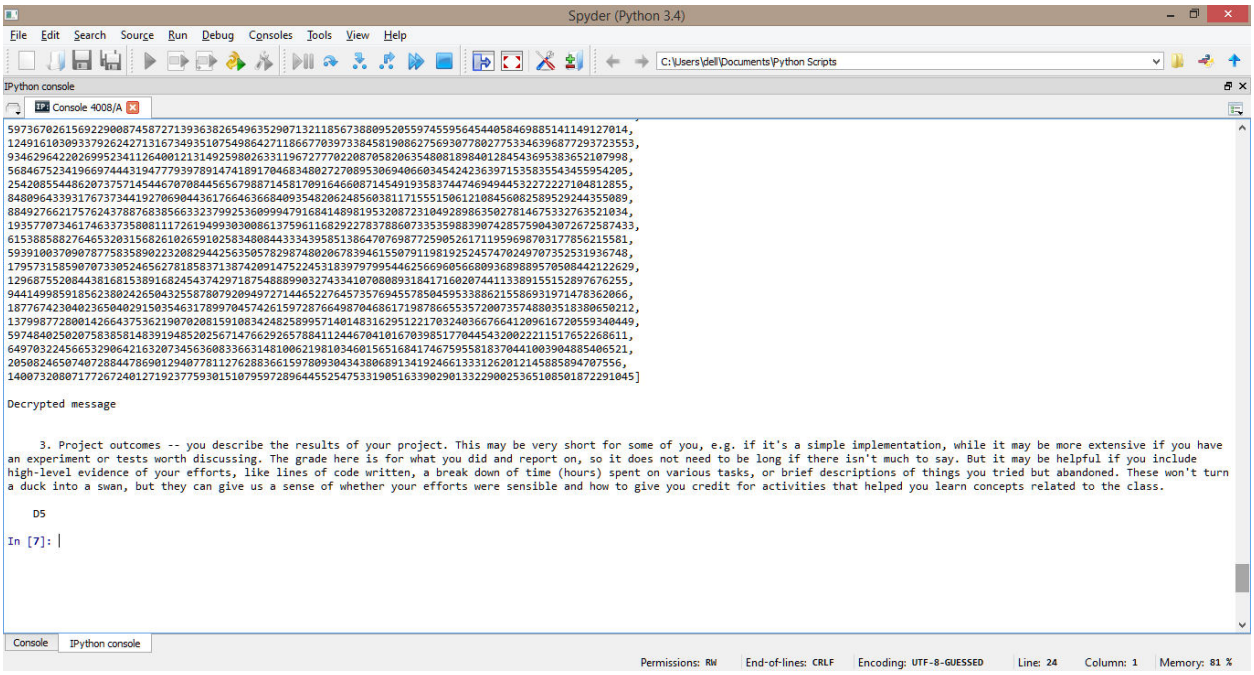
Image before embedding cipher text

Image after extracting cipher text

Cipher text extracted from the image which goes into the RSA decryption



Decrypted message using cipher text extracted from image.



Using White image for embedding to see the change:

Image before embedding same message as above

Image after extracting-You can see a wide rectangular patch at the center if you tilt the image. This is because the picture is of unique color and embedding is continuous. Use of pixel with continuous RGB pattern should never be used in steganography

References:

1) http://www.developer.com/java/ent/article.php/3530866/Steganography-101-using-Java.htm
2) http://en.wikipedia.org/wiki/Steganography
3) http://en.wikipedia.org/wiki/RGBA_color_space
4) http://www.guinnessworldrecords.com/world-records/first-use-of-steganography
5) http://waprogramming.com/download.php?download=50af5d0d652e21.01234314.pdf
6) http://www.steganosaur.us/
7) http://www.polestarltd.com/ttg/isspeeches/051403/index.html
8) http://en.wikipedia.org/wiki/HSL_and_HSV