Secure data encryption using Image Steganography
IE 509 Course Project Report

T Karthik Rajendran
23N0459

November 2023

Introduction

Sometimes information can be sensitive. While sharing sensitive information, we want to ensure that only the target recipient will know what we are sharing. Steganography is the art or practice of concealing a message, image, or file within another message, image, or file. Image Steganography involves hiding information within image files.

Types of Steganography

There are basically three Steganography types:

- Pure Steganography
- Secret key Steganography
- Public key Steganography

Pure Steganography is a Steganography system that doesn't require prior exchange of some secret information (like a key) before sending a hidden message.

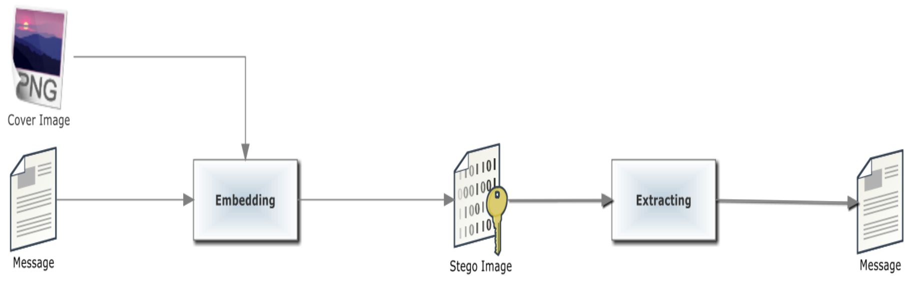


Figure 1: Pure Steganography

Libraries and methods used

I have used the PIL (Python Imaging Library) package for image processing by importing the image module. [1]

1. I have used the *Image.open( )* method to open images.
2. I have used the *Image.copy( )* method to create a copy of an image.
3. I have used the *Image.size( )* method to find the size of the image in pixels.
4. I have used the *Image.getdata( )* method to get the contents of an image as a sequence object containing pixel values.
5. I have used the *Image.putpixel( )* method to modify the RGB intensity of the pixel at a given position.
6. I have used the *Image.save( )* method to save an image under a given filename.

Encryption using the *encode( )* function

1. In encryption, we accept an image (a PNG file), a secret text message and name of the new image as input from the user.
2. We create a copy of the image.
3. We convert each character in the message into its ASCII value and write the ASCII value in base-2 using 8 bits (with leading zeroes) using the *convert_into_8_binary_bits( )* and *convert_into_ASCII( )* functions.
4. For each character in the message, we need 3 pixels. We iterate over the pixels in the copy of the image from left to right, grouping 3 pixels together at a time, containing a total of 9 colour values as every pixel has 3 colour values indicating the RGB intensities.

5. Out of these, we modify the first 8 colour values to store the ASCII value of the character in base-2. The new colour value is made odd if the corresponding bit of the ASCII value of the character in base-2 is 1 and the new colour value is made even if the corresponding bit of the ASCII value of the character in base-2 is 0. We do this by letting the new colour value be the original colour value or (original colour value - 1).
6. The 9th colour value is used to indicate the end of the message. It is made even if we are not at the end of the message and odd at the end of the message.
7. In the previous steps, we only modified the copy of the image. Now, we rename the copy of the image to the name of the new image provided by the user and save it as a PNG file. This is the output image.

I have mentioned the general idea behind the encryption algorithm I have used but the execution is slightly different. Instead of modifying pixels in the copy of the image after reading each character, I append what the colour values of the modified pixels should be to a list after reading each character using the *modPix( )* function. When I have reached the end of the message, I modify the corresponding pixels in the copy of the image based on this list using the *encode_info( )* function.

Decryption using the *decode( )* function

1. In decryption, we accept an image (a PNG file) with the hidden secret text message (after encryption as described above) as input from the user.
2. We iterate over the pixels in the image from left to right, grouping 3 pixels together at a time, containing a total of 9 colour values as every pixel has 3 colour values indicating the RGB intensities.
3. Out of these, we use the first 8 colour values to get the ASCII value of the character in base-2. The binary bit is 1 if the corresponding colour value is odd and the binary bit is 0 if the corresponding colour value is even.
4. We convert the ASCII value from base-2 to base-10 and find the character.
5. If the 9th colour value is even, we continue this procedure. Else, we stop and display the message because we are at the end of the message.

Implemented System

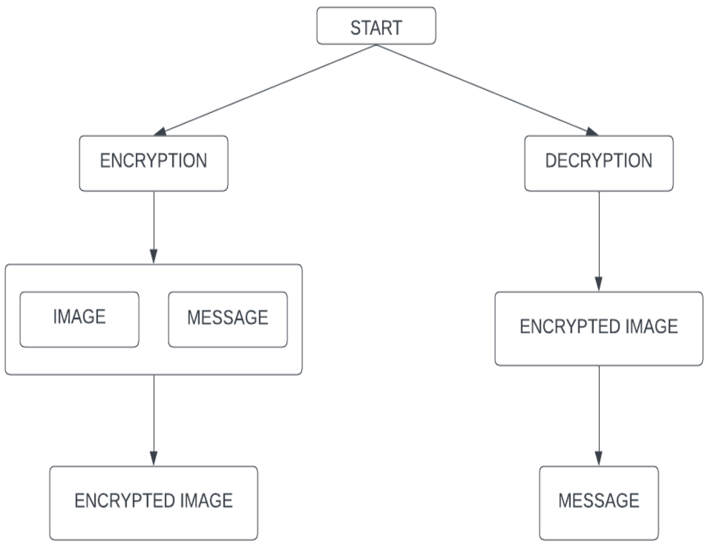
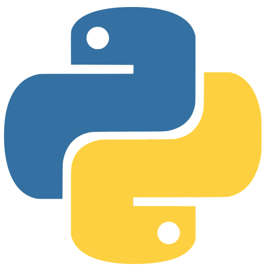


Figure 2: Flow Diagram

Example


(a) Original Image

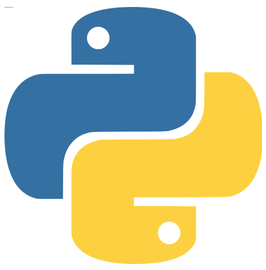

(b) Encrypted Image

Figure 3: Original Image and Encrypted Image for secret text message "Hello, World!"

Conclusion

Steganography has many real life applications and we have just seen an example of Image Steganography. We can now hide a secret text message inside an image in PNG format using the encryption algorithm and get back the message using the decryption algorithm. The parity coding algorithm I have implemented is very commonly used. [2]

The code, final conclusions and analysis in my project are all my own.

References

[1] Image Module in Pillow (PIL Fork) 10.1.0 documentation
https://pillow.readthedocs.io/en/stable/reference/Image.html

[2] Image based Steganography using Python
https://www.geeksforgeeks.org/image-based-steganography-using-python/