Course Project Report

Report submitted in fulfillment of the requirements
for the Network Security Project of

Fourth Year IDD

by

Arya Haldar - 17074004

Dhruv Gupta - 17074006

Dhruv Ajit Mahajan - 17074007

Josephine Crystal R Mathew - 17074017

Under the guidance of

Dr. Kaushal Kumar Shukla

Department of Computer Science and Engineering
INDIAN INSTITUTE OF TECHNOLOGY (BHU) VARANASI
April 2021

Declaration

I declare that

1. The work contained in this report is original and has been done by us only.
2. The work done by us in this project has not been submitted anywhere for any project or research.

Place: IIT (BHU) Varanasi
Date: 21/04/2021

Project Members
IDD Part IV
Department of Computer Science and Engineering,
Indian Institute of Technology (BHU) Varanasi,
Varanasi, INDIA 221005.

Contents

List of Figures

Abstract

Steganography is the practice of concealing a secret message behind a normal message. It stems from two Greek words, which are *steganos*, means covered and *graphia*, means writing. Steganography is an ancient practice, being practiced in various forms for thousands of years to keep communications private. Be it hiding secret text in paintings, hiding invisible message on object or leveraging clothes stitching to hide a message.

In this project,we implement public-key steganography where we start with hiding raw strings into an image and then extending the algorithm to accommodate hiding any file into an image. The input data is firstly encrypted by AES-128 and then using Least-Significant-Bit pixel encoding, the input data is hidden within the image. The information retrieval is done from the image using the reverse logic. The whole algorithm is implemented on the basis that the resultant image formed should not differ significantly from the original.

Chapter 1

Steganography

1.1 Objective

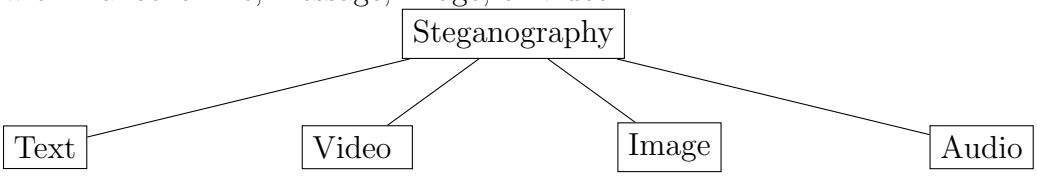
In this project, we aim to hide raw string and multiple files inside an image of suitable size. The objective is to create an encoded image that consists of encrypted information as well as keeping the original image as intact as possible.

1.2 Introduction

The term steganography refers to a technique that aims to hide communication between two interlocutors. The term is composed precisely of the Greek words *steganos*, meaning covered and *graphia* meaning writing, Unlike encryption, which allows you to encrypt a message so as to make it incomprehensible if you do not have a key to decipher it, steganography aims to keep the very existence of the message away from prying eyes, by hiding it.

Steganography is the practice of concealing a message within another message or a physical object. For eg. Hiding a secret message inside a piece of art. or another example, using a special invisible ink to write content over an image. In computing/electronic contexts, a computer file, message, image, or video is concealed

within another file, message, image, or video.



1.3 Theory

1.3.1 AES-128

AES encryption is established by the United States National Institute of Standards and Technology (NIST) in 2001 and it aims to offer a specification for the electronic data encryptionAES encryption has three different block ciphers: AES-128 (128 bit), AES-192 (192 bit) and AES-256 (256 bit). These block ciphers are named after the key length they use for encryption and decryption. All these ciphers encrypt and decrypt the data in 128-bit blocks but they use different sizes of cryptographic keys. AES is characterized as being a symmetric block cipher, in other words it uses the same key for encryption and decryption. In this project, we use AES to encode the raw input.It is done because the data to be hidden in the image has to be encrypted using a strong symmetric encryption scheme and it has to be randomly hidden to prevent LSB decoders from detecting the hidden data.

1.3.2 Least significant bit pixel encoding

Digital images are mainly of two types (i) 24 bit images and (ii) 8 bit images. In 24 bit images we can embed three bits of information in each pixel, one in each LSB position of the three eight bit values. Increasing or decreasing the value by changing the LSB does not change the appearance of the image; much so the resultant stego image looks almost same as the cover image. In 8 bit images, one bit of information can be hidden. If the LSB of the pixel value of cover image C(i,j) is equal to the message bit m of secret message to be embedded, C(i,j) remain unchanged; if not, set

the LSB of C(i, j) to m. The message embedding procedure is given below:

1. S(i,j) = C(i,j) - 1, if LSB(C(i,j)) = 1 and m = 0
2. S(i.j) = C(i,j), if LSB(C(i,j)) = m
3. S(i,j) = C(i,j) + 1, if LSB(C(i,j)) = 0 and m = 1

where LSB(C(i, j)) stands for the LSB of cover image C(i,j) and m is the next message bit to be embedded. and S(i,j) is the stego image.

For example, suppose one can hide a message in three pixels of an image (24-bit colors). Suppose the original 3 pixels are:
(11101010 11101000 11001011) (01100110 11001010 11101000) (11001001 00100101 11101001)

A steganographic program could hide the letter "J" which has a position 74 into ASCII character set and have a binary representation "01001010", by altering the channel bits of pixels. The last bit remains unchanged.

(11101010 11101001 11001010) (01100110 11001011 11101000) (11001001 00100100 11101001)

In this case, only four bits needed to be changed to insert the character successfully. The resulting changes that are made to the least significant bits are too small to be recognised by the human eye, so the message is effectively hidden. The advantage of LSB embedding is its simplicity and many techniques use these methods.

Pixel Value	Decimal Value	Binary Value	Encoded Binary Value	Encoded Decimal Value
Red	255	11111111	11111111	255
Green	240	11110000	11110001	241
Blue	212	11010100	11010101	213

Figure 1.1 Encoding "111" inside 1 pixel using RGB LSB data hiding

1.4 Encoding Algorithm

The program takes in four inputs from the user:

1. Data (plain strings or file(s) of any type)
2. Secret key for the AES encryption and random number generation
3. Source PNG image
4. Destination PNG image name

The secret key is fed into the SHA-256 hashing algorithm which outputs a 256-bit hash. This 256-bit hash is used for three purposes:

1. The first 128 bits of the hash are used for the initialization vector for AES
2. The second 128 bits of the hash are used as the secret key for AES (Note: AES-128 is being used to encrypt the data)
3. The entire 256-bit hash is used as the seed for the random number generator.

Without the secret key, retrieving the hidden data is not possible. It is imperative for the receiving party to know the secret key in order to retrieve the hidden data

inside it. The random number generator takes in another input, the total number of encodable bits, which is calculated by multiplying the length of the image by the width of the image and subtracting 1 row (the length) from the product. This is necessary for creating a random number (the hop number) of the correct size, where each pixel needs to be represented by one bit of the random number (Note: These random numbers are extremely large).

For example, if the total number of encodable pixels is 100, a 100-bit random number would be generated using the 256-bit hash of the secret key as a seed for the random number generator. The first row (row 0) is subtracted from the total number of pixels in the image to get the total number of encodable pixels because the first row is used for the header.

The header comprises two values:

1. The first 10 pixels of the image contain the length of the ciphertext in binary encoded in the LSB of the RGB values

2. The zip file header (if the user-specified a file or files as the data to hide).

The random number, now converted into binary, is the key to hiding the encrypted data randomly. Starting at the first pixel in row 1 (the image is represented as a 2-D array which is 0 bounded for both the rows and columns) and working through the image. We encode 3 bits of ciphertext in each pixel that maps to a 1 in the random number. If a pixel maps to a 0, that pixel is not encoded with any data and is skipped (initially). Once the encoding has finished, the resulting image is saved as per the user-specified name in the current working directory

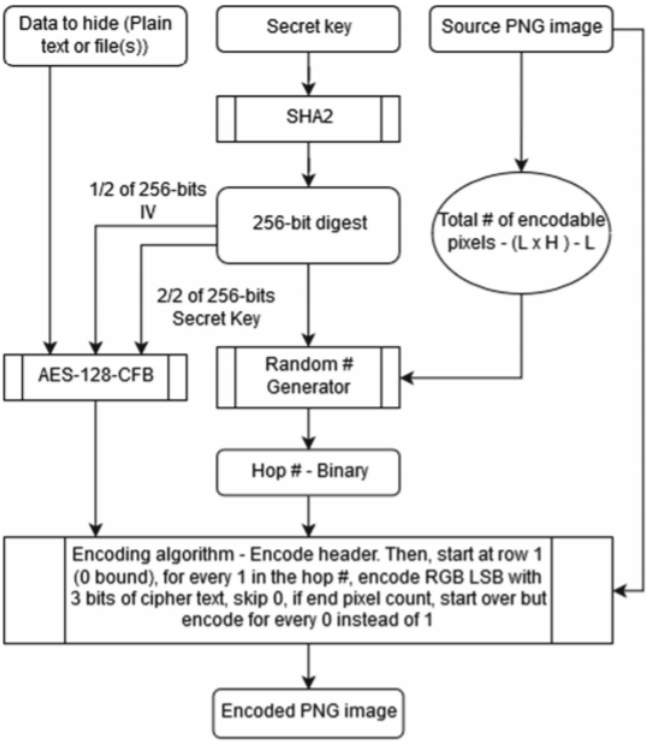


Figure 1.2 Flowchart of the encoding algorithm

1.5 Decoding Algorithm

Decoding algorithm is essentially the opposite of encoding. Two inputs are taken from the user for decoding function: the secret key and the encoded image which has data hidden in it. The algorithm can understand whether the encoded image contains files or text. It knows this by looking at the second part of header information stored in row 0 of image.

If it matches the zip file header, then the decoding algorithm knows hidden data contains one or more files. The secret key given by the user is used to retrieve the hidden data. It is again provided to the SHA-256 bit hashing algorithm. This produces a 256 bit hash, with the first 128 bits being used as initialization vector and the second 128 bits as secret key for AES decryption. (AES is a symmetric key algorithm so this secret key is used for encryption as well)

It starts at the first pixel in row 1, and it reads 3 bits of cipher text from each pixel that maps to a 1 in the random number. It does this by reading LSB of RGB values of such pixels. If after reaching the end of the image, some cipher text is left to be found, it goes to the beginning of the image and starts reading pixels that map to a 0 in the random number.

How much cipher text is left to be decoded is found according to the cipher text size stored in the image in the first part of the header. Once all the cipher text is read, it is sent to the AES decryptor. Thus, the random number helps to retrieve the hidden data in a correct sequence.

If the image header includes the zip file header, the decoding algorithm writes the decrypted bytes as a zip archive. This zip archive is extracted into a HIDDEN_DATA folder.

1.6 Implementation

There are some assumptions in the implementation of this project:

1. The image format where the data is to be hidden inside is of PNG format. JPG, JPEG formats have a lossy compression format, which means some of the detail of your image will be lost when saved in order to keep a low file size. Since we are using zip compression, data might be lost. This makes it impossible for us to recover the original data.

2. The image should have sufficient size to hide the input data.

We created a command line interface to accommodate taking raw string as input, multiple files as input and to retrieve data from the image.

1.6.1 Hiding Raw String into an Image

```
Windows PowerShell
PS C:\Users\C HALDAR\Desktop\Sem8\NS\Project\Steganography> python stegano.py
Select a specific functionality from the menu below

1) Hide a secret message into an image
2) Find a secret message from an image
3) Exit

Menu option selection -> 1
Select a specific functionality from the menu below

1) Hide a raw string
2) Hide a single file, or multiple files
3) Go back

Menu option selection -> 1

Enter a SECRET MESSAGE to hide -> Hello World
Enter a SECRET KEY to encrypt the secret message -> 123455
Provide the PATH of the source image -> shuttle.png
Provide the NAME for the encoded image -> s2.png

Encoding...



########################################################################
DONE -- MODIFIED image saved to current directory as: s2.png
########################################################################

Select a specific functionality from the menu below

1) Hide a secret message into an image
2) Find a secret message from an image
3) Exit

Menu option selection ->
```

Figure 1.3 Before Encoding of Raw String

Figure 1.4 Input Image

Figure 1.5 Output Image

The images dont seem different, however if they are passed through SHA-256, we get two different outputs.

Input Image SHA256: **9b03df019b72...**

Output Image SHA256: **71de2439a46c....**

1.6.2 Finding Raw String from an Image

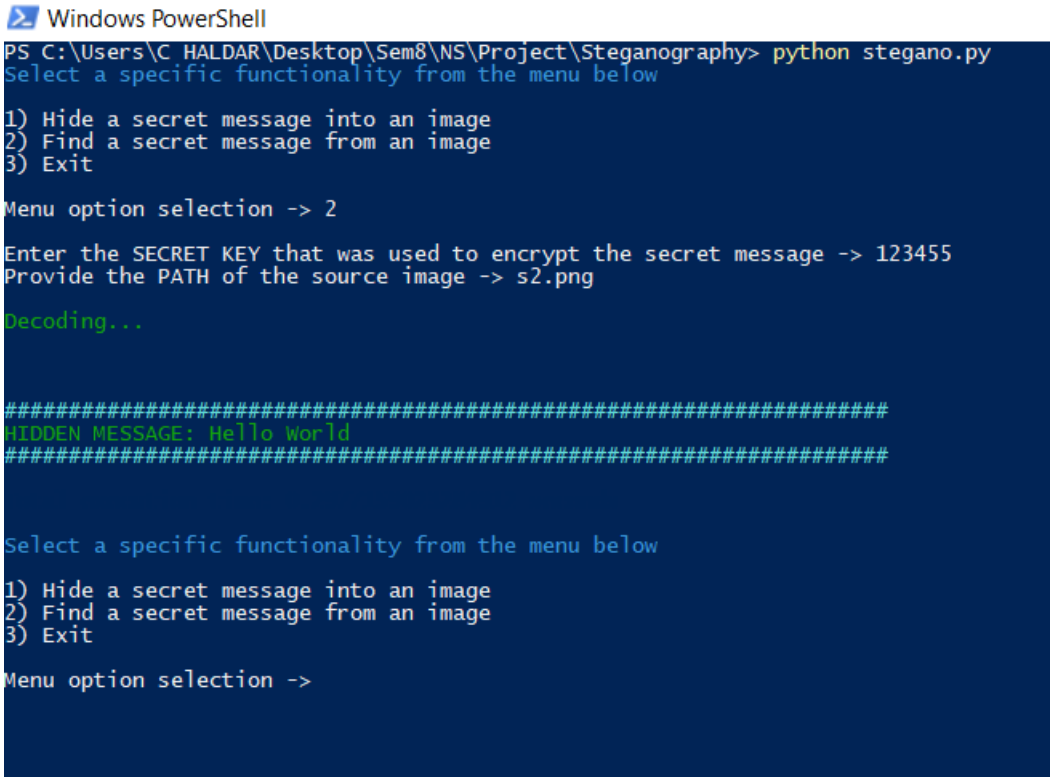


Figure 1.6 Retrieving Raw String

1.6.3 Hiding Multiple files into an Image

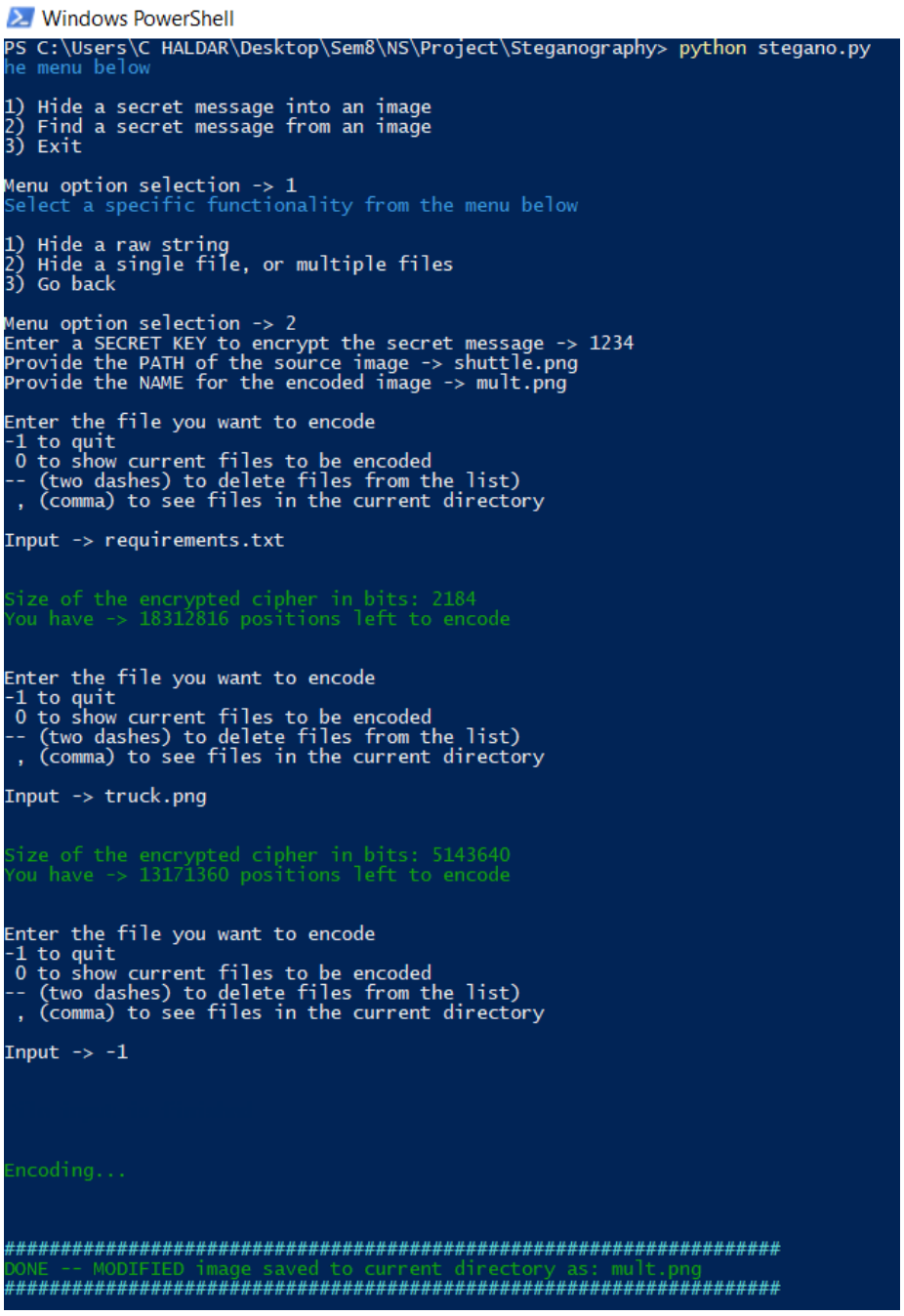


Figure 1.7 Hiding multiple files

1.6.4 Retrieving Files from an image

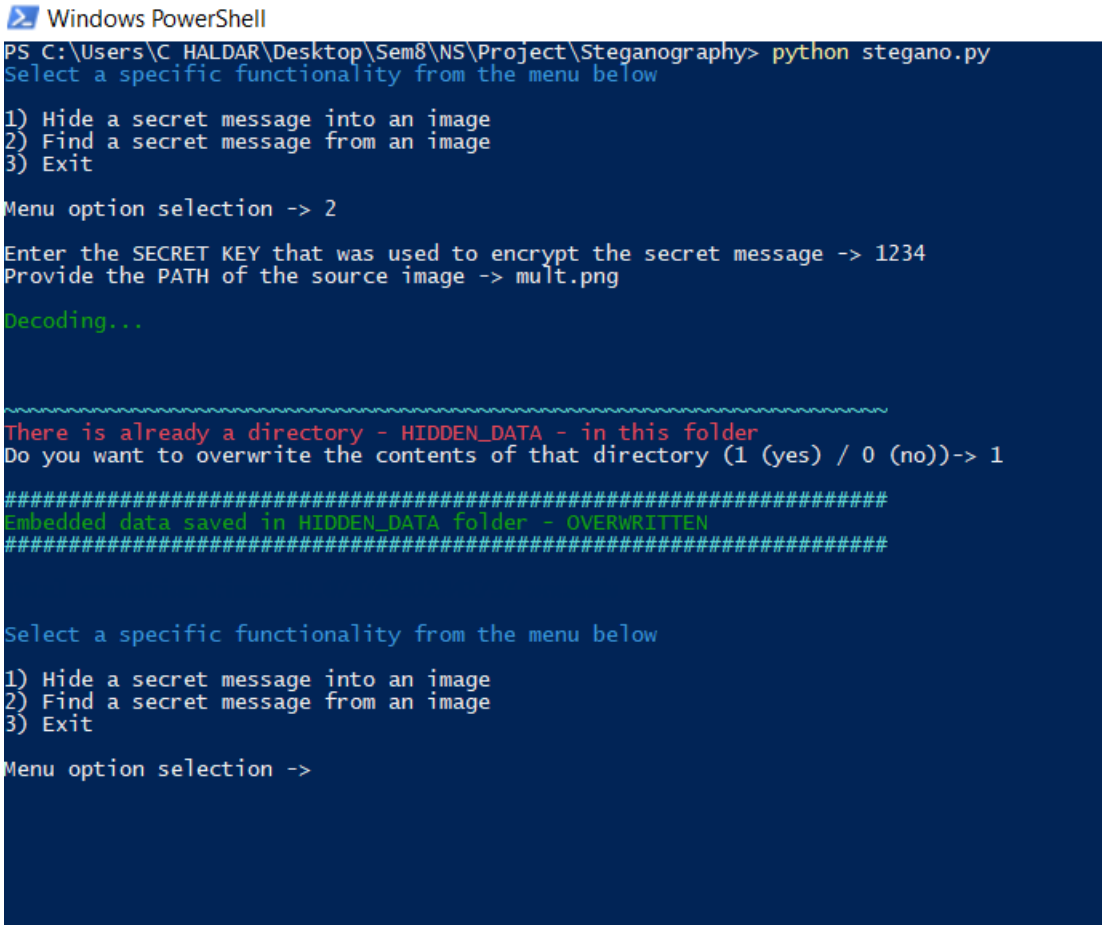


Figure 1.8 Retrieving Multiple files

Figure 1.9 Resultant Folder

1.7 Code

The code for this project can be found **here**

1.8 Contribution of Participating members

Member	Contribution
Arya Haldar	UI creation and extension to multiple files
Dhruv Gupta	Data hiding algorithm and Research
Josephine Crystal Mathew	Data Encoding
Dhruva Mahajan	Data Decoding

1.9 Conclusion

In this project, we used the LSB data hiding method combined with a random pixel encoding algorithm to distribute the data throughout the image. Additionally, AES encryption is deployed against the data to encrypt it before it is hidden inside the image. For 24-bit images, where each pixel has an 8-bit R, G, and B value, 3 bits of data can be stored in the LSB of every pixel. By working with the image as a 2-D array, the image is able to be manipulated using standard array manipulation techniques.

We use the secret key's SHA-256 hash as a seed for the random number generator, a unique random number is created that is used to hide the encrypted data throughout the image by encoding data at pixels which correspond to a 1 in the random number's binary representation.

Multiple files of any type are supported by using a zip archive to combine all the files under one file type, so we only has to deal with one type of file header instead of

the literal hundreds of different file types. The decoding function works in a similar way to the encoding function, except instead of manipulating the LSB, it reads them, decrypts them, and then based on the image header, creates a folder to hold all the hidden files or displays the hidden string.