Final Project - Steganography

Sukh Atwal
Yoshiaki Ryuzaki

Contents

Introduction

Objective

The objective of the final project is to familiarize ourselves with steganography and how it works through designing and implementing an LSB steganography application. We've created an executable application with a GUI that allows you to encrypt as well as decrypt images through the actual GUI or through a command line.

Instead of limiting to just BMP picture formats, you're allowed to encrypt and decrypt any other image format as well as long it's supported by the OpenCV library. This includes png, jpeg, jpg, tiffs and many others.

You can view the full lists on the OpenCV documentation at:

http://opencv.jp/opencv-2.2_org/cpp/highgui_reading_and_writing_images_and_video.html

Libraries used:

The application is built using Python 3 (3.7).

Here is a list of the libraries used by the application:

- Tkinter – Used to build up the GUI
- OpenCV – Used for processing the images
- NumPy – Used for manipulating the image arrays
- Pillow – Used with Tkinter for loading the logo

The executable application runs standalone and does not depend on any of the libraries.

But if you wish to use the CLI with the source codes, you do need to install above libraries. There is a requirements.txt file in the source code folder. Make sure you have python3 and pip installed. Use the following command to install all of the required libraries:

```
pip install -r requirements.txt OR
pip3 install -r requirements.txt
```

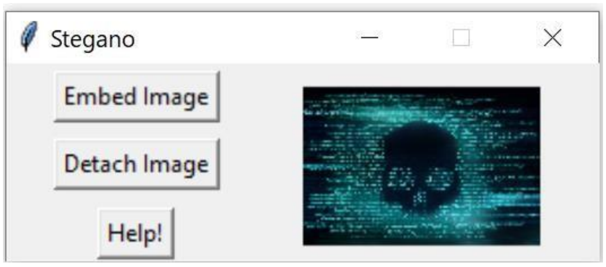
How to Use

The program supports both GUI and CLI interactions.

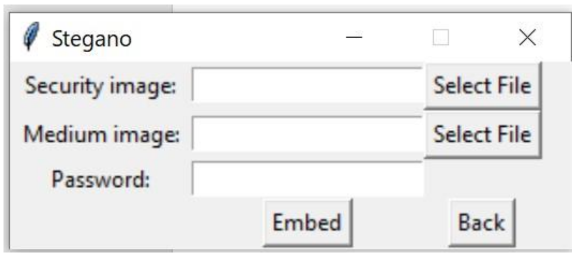
GUI:

To use GUI there are two options:

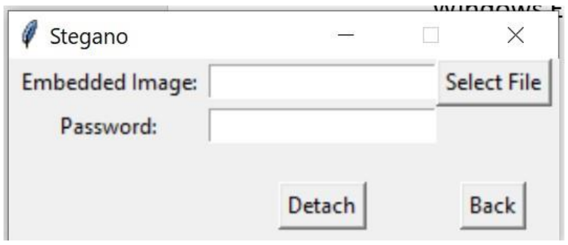
1. Open GUI using the executable file:
   Windows Environment:
      - ✦ Navigate to the executable folder, Double Click stegano.exe ✦ Choose either the embed or detach image mode:



- ✦ If embed image mode is chosen:
    - ○ Choose the Security Image (Secret Image)
    - ○ Choose the Media Image ○ Enter the Password to use ○ Click Embed



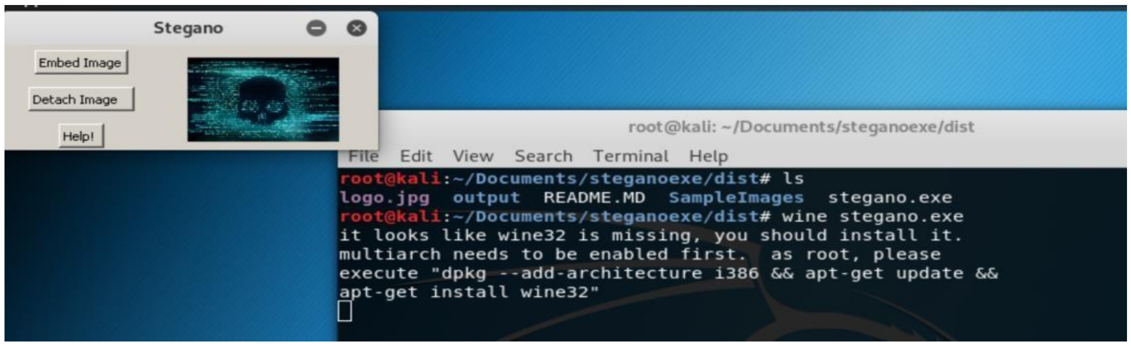
- ○ The Embedded (Media) Image will pop up
- ○ Close the popped-up image, and check the output folder ○ The Embedded (Media) Image should be saved to there ✦ If Detach Image mode is chosen:
- ○ Choose the Embedded (Media) Image ○ Enter the right password



- ○ Click Detach
- ○ The Secret Image will pop up, if the password is incorrect nothing will be shown ○ Close the popped-up image, and check the output folder
- ○ The Secret Image with original filename and extension should be saved to there

Linux Environment:

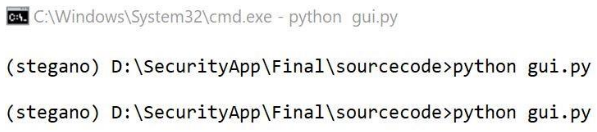
- ✦ Navigate to the program folder
- ✦ Use command “wine stegano.exe”


Stegano
Embed Image
Detach Image
Help!
root@kali: ~/Documents/steganoexe/dist
File Edit View Search Terminal Help

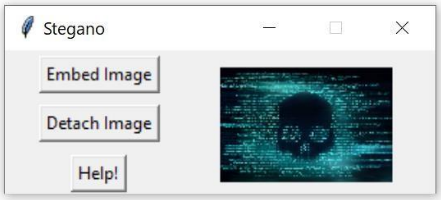
```
root@kali:~/Documents/steganoexe/dist# ls
logo.jpg  output  README.MD  SampleImages  stegano.exe
root@kali:~/Documents/steganoexe/dist# wine stegano.exe
it looks like wine32 is missing, you should install it.
multiarch needs to be enabled first.  as root, please
execute "dpkg --add-architecture i386 && apt-get update &&
apt-get install wine32"
```

- ✦ The rest will be same with the windows steps

2. Open GUI with the source code
   Any environment
   - Open Command Prompt (Or terminal)
   - Navigate to the source code folder using Command Prompt (Or terminal)
   - Run "python gui.py" (Make sure use python here, otherwise some libraries may not load correctly)
   - The rest of the steps will be same with the Open GUI using the executable file



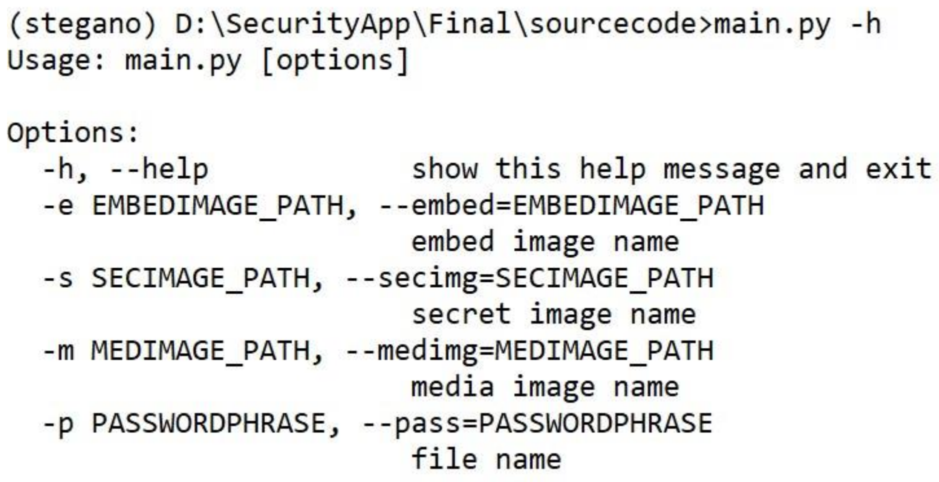
```
C:\Windows\System32\cmd.exe - python  gui.py

(stegano) D:\SecurityApp\Final\sourcecode>python gui.py

(stegano) D:\SecurityApp\Final\sourcecode>python gui.py
```



CLI:

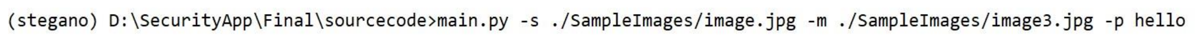
To use CLI in any environment:

- Open Command Prompt (Or terminal)
- Navigate to the source code folder
- Check all the options with main.py -h



```
(stegano) D:\SecurityApp\Final\sourcecode>main.py -h
Usage: main.py [options]

Options:
  -h, --help            show this help message and exit
  -e EMBEDIMAGE_PATH, --embed=EMBEDIMAGE_PATH
                        embed image name
  -s SECIMAGE_PATH, --secimg=SECIMAGE_PATH
                        secret image name
  -m MEDIMAGE_PATH, --medimg=MEDIMAGE_PATH
                        media image name
  -p PASSWORDPHRASE, --pass=PASSWORDPHRASE
                        file name
```

- To embed a Secret image to the Media image, use -s, -m and -p flags
  Command: main.py -s < secret image> -m < media image> -p <passphrase>

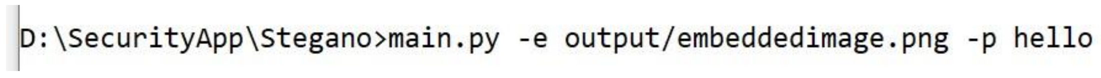


```
(stegano) D:\SecurityApp\Final\sourcecode>main.py -s ./SampleImages/image.jpg -m ./SampleImages/image3.jpg -p hello
```

○ The Embedded (Media) Image will pop up

- ○ Close the popped-up image, and check the output folder ○ The Embedded (Media) Image should be saved to there

✦ To detach a Secret image from the Embedded (Media) Image, use -e, -p flags
Command: main.py -e < embedded image> -p <passphrase>



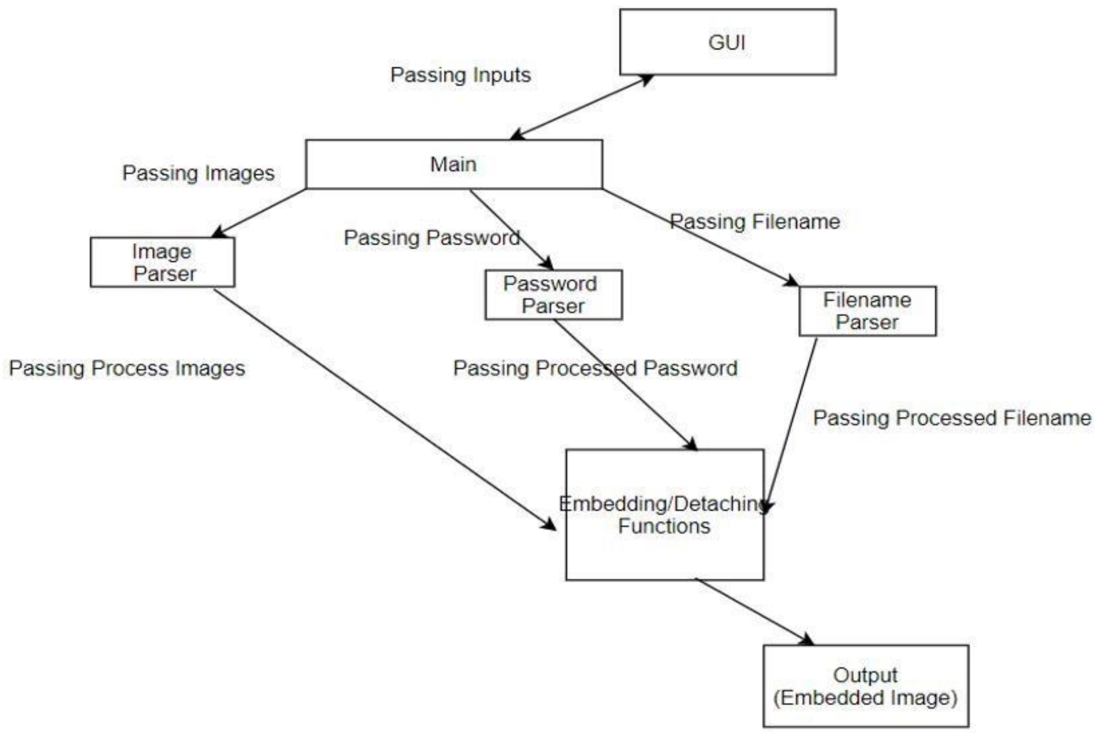
```
D:\SecurityApp\Stegano>main.py -e output/embeddedimage.png -p hello
```

○ The Secret Image will pop up, if the password is incorrect nothing will be shown ○ Close the popped-up image, and check the output folder

- ○ The Secret Image with original filename and extension should be saved to there

Design & Features

Program Functions

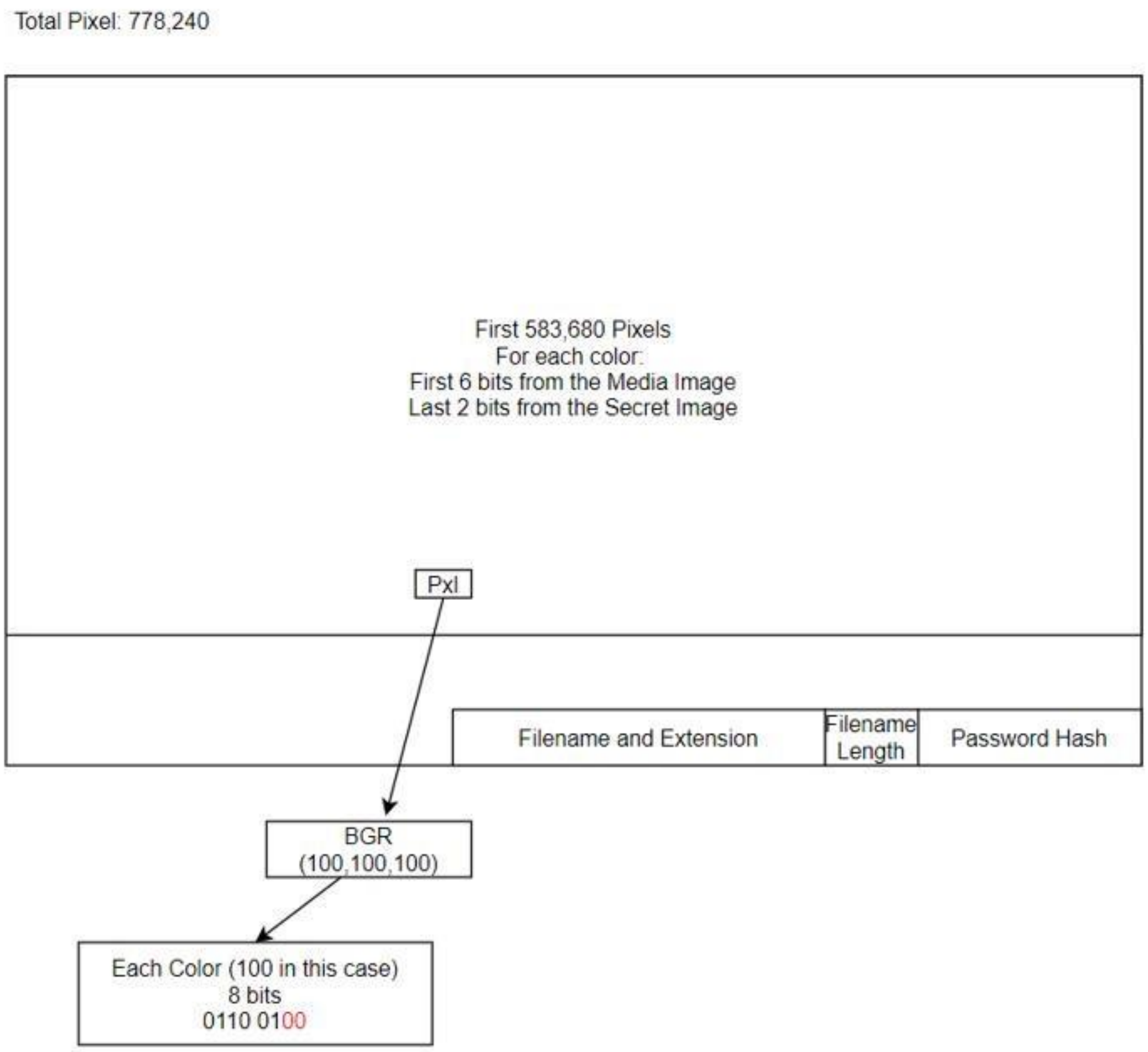


The program includes 6 main modules, each module includes multiple functions:

1. GUI – The GUI is the module that directly interacts with users. It is also the entry point of the program. Users will pass inputs through the GUI module.
2. Main – The Main module is the center of the program. It connects to all the modules and decides what functions to execute. It is also the entry point of the CLI option.

3. Image Parser – The Image Parser includes all the functions related to general image processing with functions such as loading, resizing and showing images.
4. Password Parser – Password Parser includes modules related to passphrase processing such as hashing and hash to binary array conversion.
5. Filename Parser – The Filename parser function includes functions like getting the filename and filename to binary array conversion.
6. Embedding/ Detaching – The Embedding/Detaching functions includes functions to embed the binary arrays to the media image using LSB, and functions to detach those arrays.

Embedded Image



Here is how an embedded (media) image looks after we have had the secret image embedded into the media image.

For the media image, a size 1024x760 image was used; 512x380 image for the secret image. The total number of pixels of the media image is 778,240 (1024 x 760).

We will use the first 583,680 pixels' BGR color values' last two bits to store the secret image's BGR color value. The reason why it's 583,680 pixels is because we divide each pixel's BGR color value of the secret image by 4 when we embed it to the media image, and then we multiply the color value by 4 when we detach the secret image from the embedded (media) image. When we divide the 8 bits color value by 4, the maximum binary value is bits long (111111) and we separate them into three 2-bit long numbers (11, 11, 11) to place them as the last 2 bits of the media image's color value.

So, in this case to hide a 512x380 (194,560) pixels image, we will have to use 194,560x3 = 583,680 pixels of the media image. The reason we divide the secret image's color value by 4 is so that we can minimize the impact on the media image by minimizing the size of the binary bits we have to insert. It also helps with making the secret image harder to be detected.

The last 21 pixels (64 color values) of the embedded (media) image will be used to store the password hash. Since MD5 hash value is always 128 bits long, we use the last 2 bits of each color value of the embedded (media) image. So, 128 / 2 / 3 = 21 pixels that will be needed.

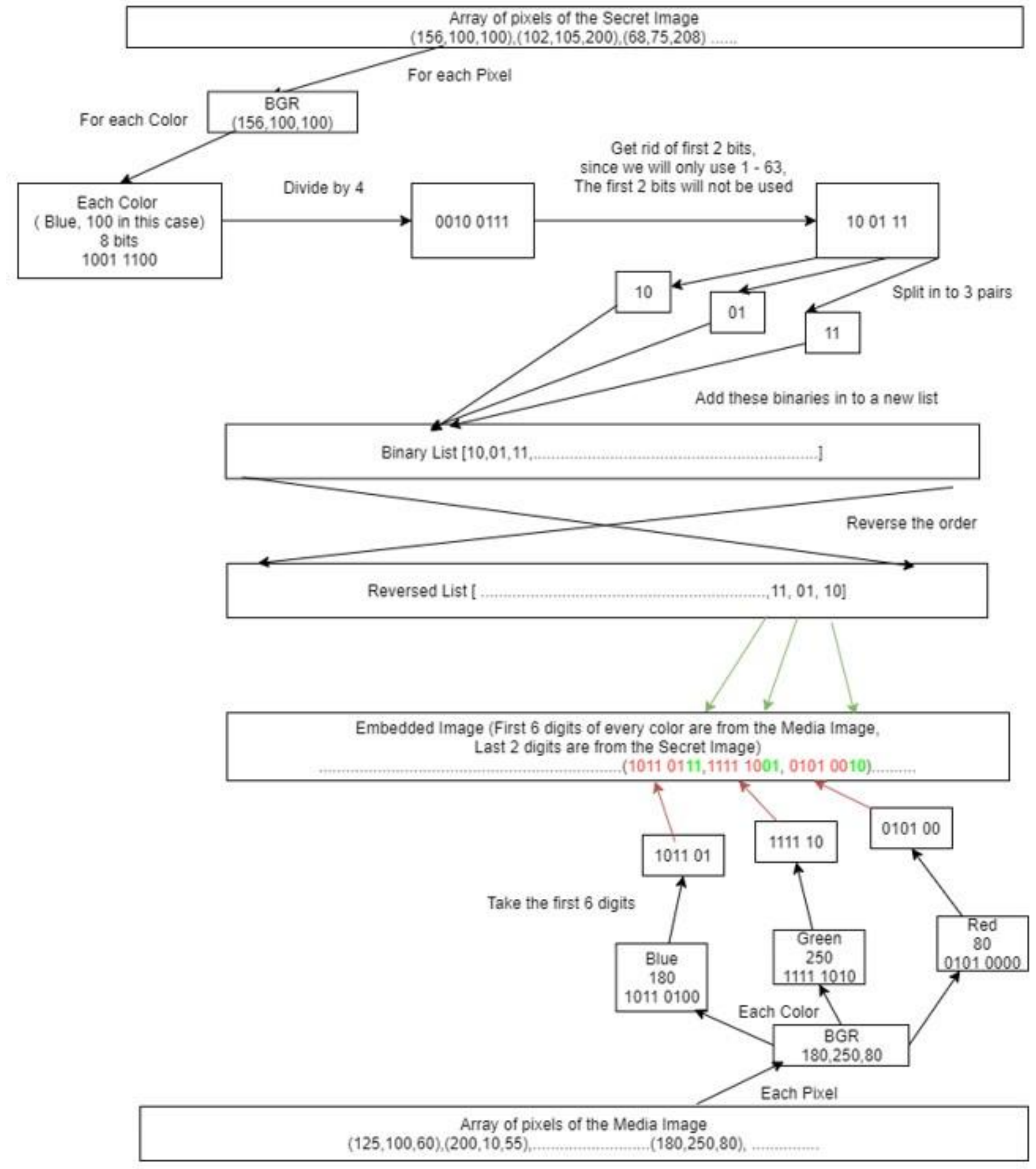
The 2 pixels right before the password hash pixels are the pixels we use to hide the length of the filename and the file extension. The length determines the range of the pixels the program should get the file name and the file extension from.

Features

Here are the designs for the image encrypting and embedding, filename embedding and password embedding features.

All three embedding features are using the same embedding method. The embedding concept is simple, the program converts the input (whether an image, a file name or a passphrase) to a binary array and inserts it into the binary array of the media image using the least significant bit (LSB). The major difference between these three features is how the program obtains the binary array and which part the media image uses to store these binaries.

Image Encrypting and Embedding



Array of pixels of the Secret Image (156,100,100),(102,105,200),(68,75,208) ......
For each Pixel
BGR (156,100,100)
For each Color
Each Color (Blue, 100 in this case) 8 bits 1001 1100
Divide by 4
0010 0111
Get rid of first 2 bits, since we will only use 1 - 63, The first 2 bits will not be used
10 01 11
Split in to 3 pairs
10
01
11
Add these binaries in to a new list
Binary List [10,01,11,..........]
Reverse the order
Reversed List [..........,11, 01, 10]
Embedded Image (First 6 digits of every color are from the Media Image, Last 2 digits are from the Secret Image)
..........(1011 0111,1111 1001, 0101 0010).........
1011 01
1111 10
0101 00
Take the first 6 digits
Blue 180 1011 0100
Green 250 1111 1010
Red 80 0101 0000
Each Color
BGR 180,250,80
Each Pixel
Array of pixels of the Media Image (125,100,60),(200,10,55),..........(180,250,80), ...

For image embedding, the program uses OpenCV library to load the secret image as a NumPy Array. Inside the array, each pixel of the secret image array consists of three sets of 0-255 numbers, which are the BGR color values of the pixel (OpenCV uses BGR instead of RGB). The program divides these numbers by 4, combines and then converts them into a binary array (Each element consists of two bits), which looks like the following:

[10, 01, 11, 00, 10, 11, 11 …..]

The reason why you divide them by 4 is to make the binary array as small as possible and to minimize the impact when the program restores the image by multiplying these numbers by 4 later on.

After the binary array conversion of the secret image, the program loads the media image and converts the image to a binary array (Each element represents one of the three BGR color value for one image pixel, consisting of eight bits). Once it finishes the binary array conversion of the media image, the program reverses the secret image binary array (which makes it far more difficult to detect) and inserts the binary numbers inside to the binary array of the media image using LSB. These binary numbers from the secret image array will replace the last two bits of the media image binary elements.

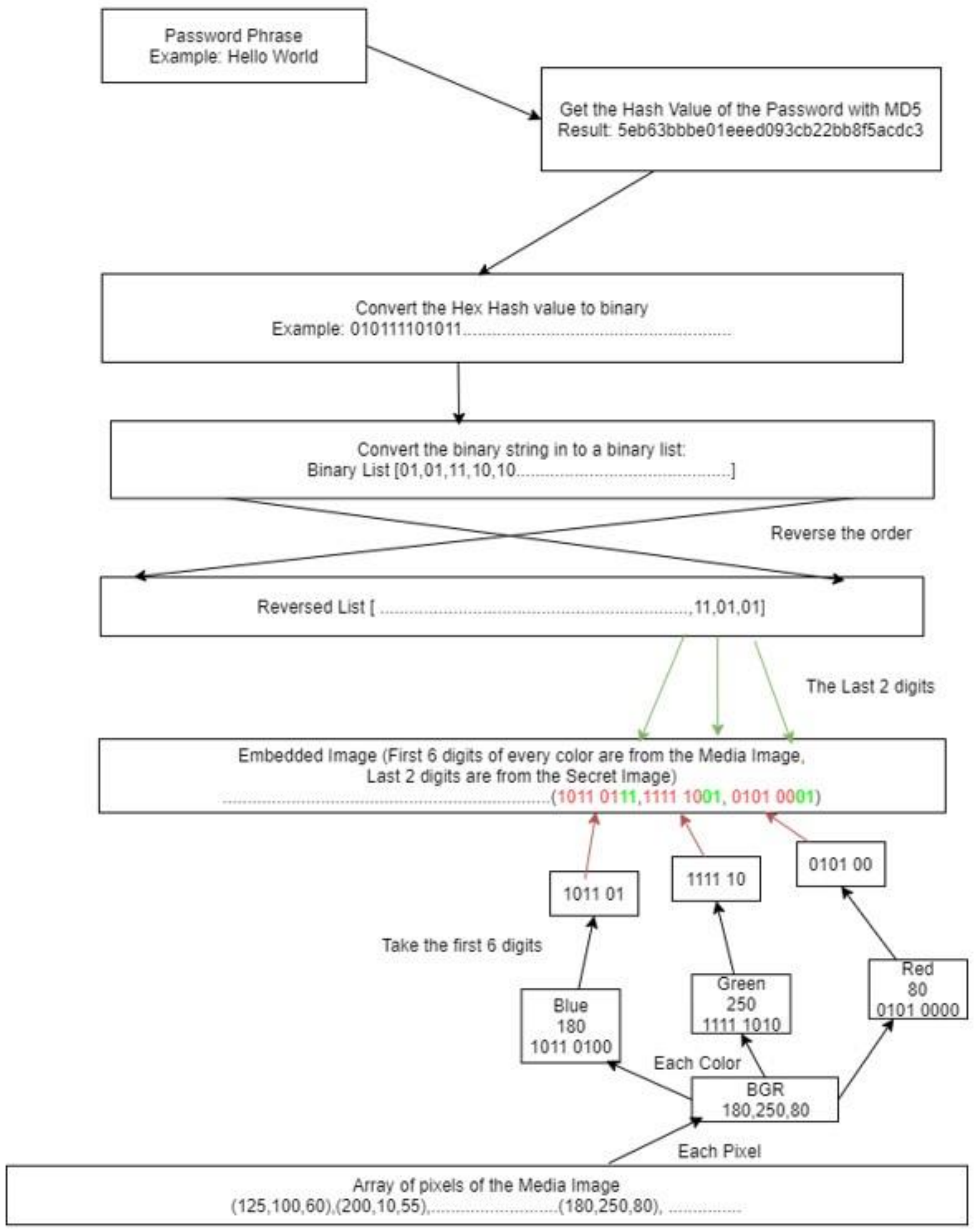
Here is an example of how the new binary array element of the media image looks like after the insertion, the red part will be the binary bits from the secret image, and the black bits are from the media image.

0011 0110

After the program has inserted all of the secret image binary bits to the media image binary array, it will convert the modified media image binary array back to an image.

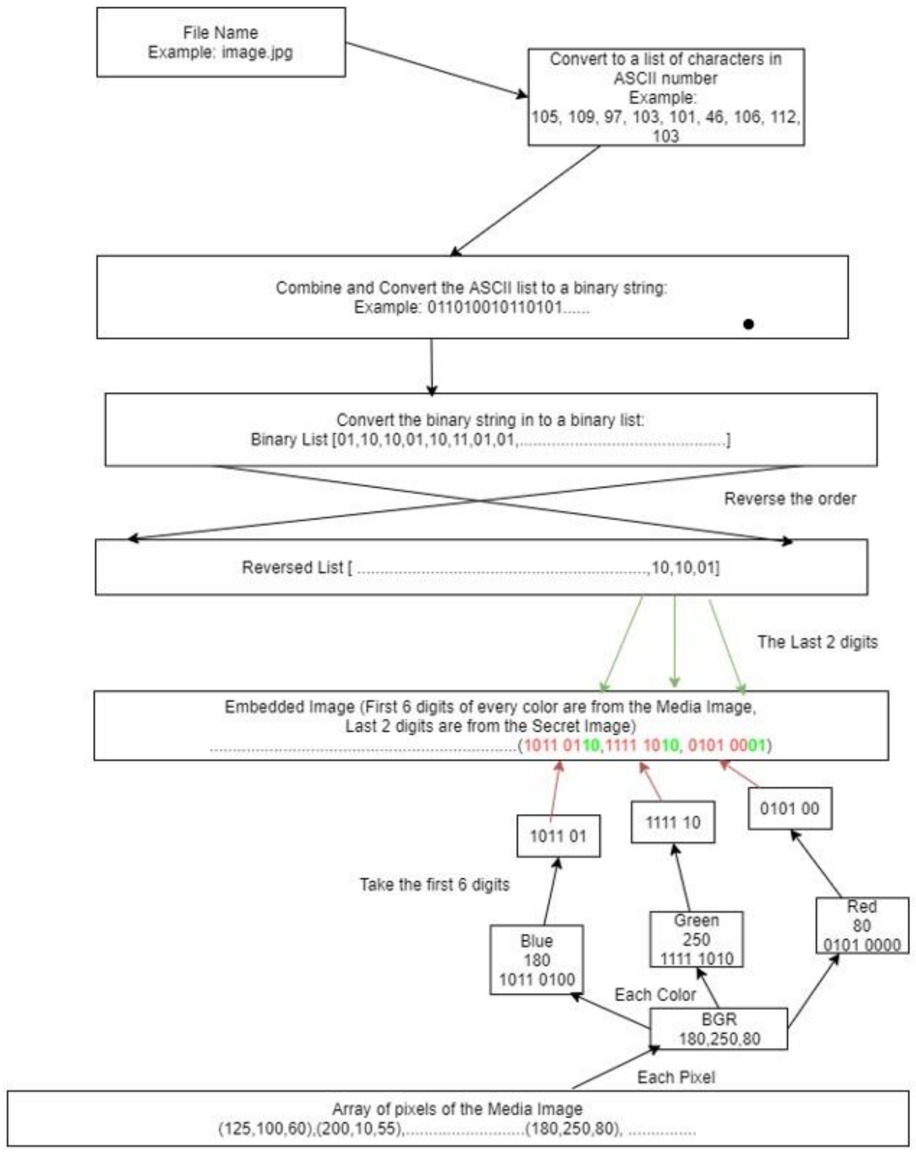
For this method only the least significant bits of the pixel colors were modified making it hard for reader to notice any signs of modifications.

Password Embedding


Password Phrase
Example: Hello World
Get the Hash Value of the Password with MD5
Result: 5eb63bbbe01eeed093cb22bb8f5acdc3
Convert the Hex Hash value to binary
Example: 010111101011..
Convert the binary string in to a binary list:
Binary List [01,01,11,10,10...]
Reverse the order
Reversed List [....,11,01,01]
The Last 2 digits
Embedded Image (First 6 digits of every color are from the Media Image, Last 2 digits are from the Secret Image)
....(1011 0111,1111 1001, 0101 0001)
1011 01
1111 10
0101 00
Take the first 6 digits
Blue
180
1011 0100
Green
250
1111 1010
Red
80
0101 0000
Each Color
BGR
180,250,80
Each Pixel
Array of pixels of the Media Image
(125,100,60),(200,10,55),..........................(180,250,80), ...

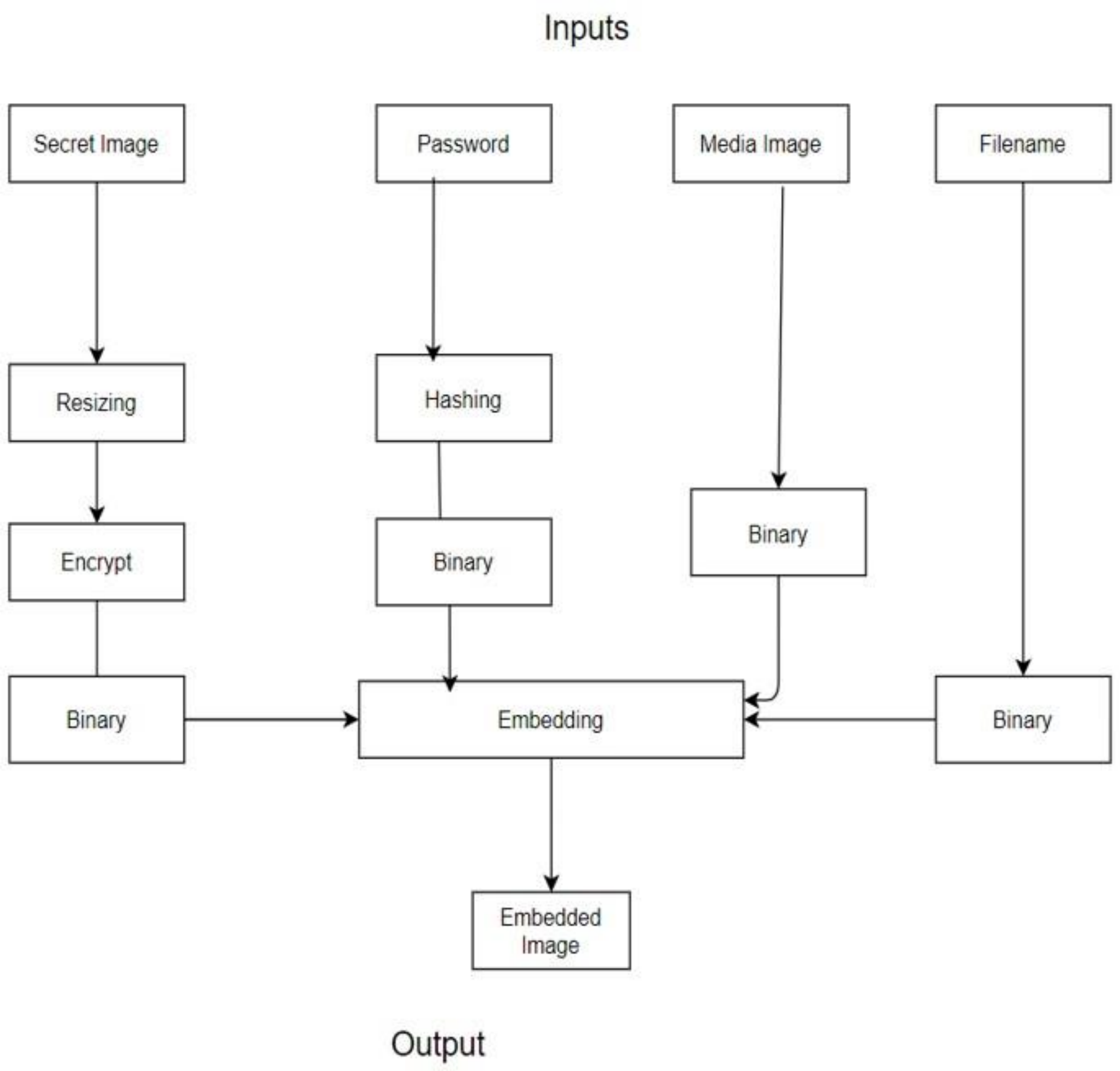
The password embedding feature acts almost the same as the image embedding feature. The major difference is that the program doesn't directly convert the input into a binary array but instead, it gets the hash values of the passphrase first and converts the hash value into a binary array. It then inserts the binary bits from the binary array to the media image.

Filename Embedding


File Name
Example: image.jpg
Convert to a list of characters in ASCII number
Example:
105, 109, 97, 103, 101, 46, 106, 112, 103
Combine and Convert the ASCII list to a binary string:
Example: 011010010110101......
Convert the binary string in to a binary list:
Binary List [01,10,10,01,10,11,01,01,..]
Reverse the order
Reversed List [....,10,10,01]
The Last 2 digits
Embedded Image (First 6 digits of every color are from the Media Image,
Last 2 digits are from the Secret Image)
....(1011 0110,1111 1010, 0101 0001)
1011 01
1111 10
0101 00
Take the first 6 digits
Blue
180
1011 0100
Green
250
1111 1010
Red
80
0101 0000
Each Color
BGR
180,250,80
Each Pixel
Array of pixels of the Media Image
(125,100,60),(200,10,55),..........................(180,250,80), ...

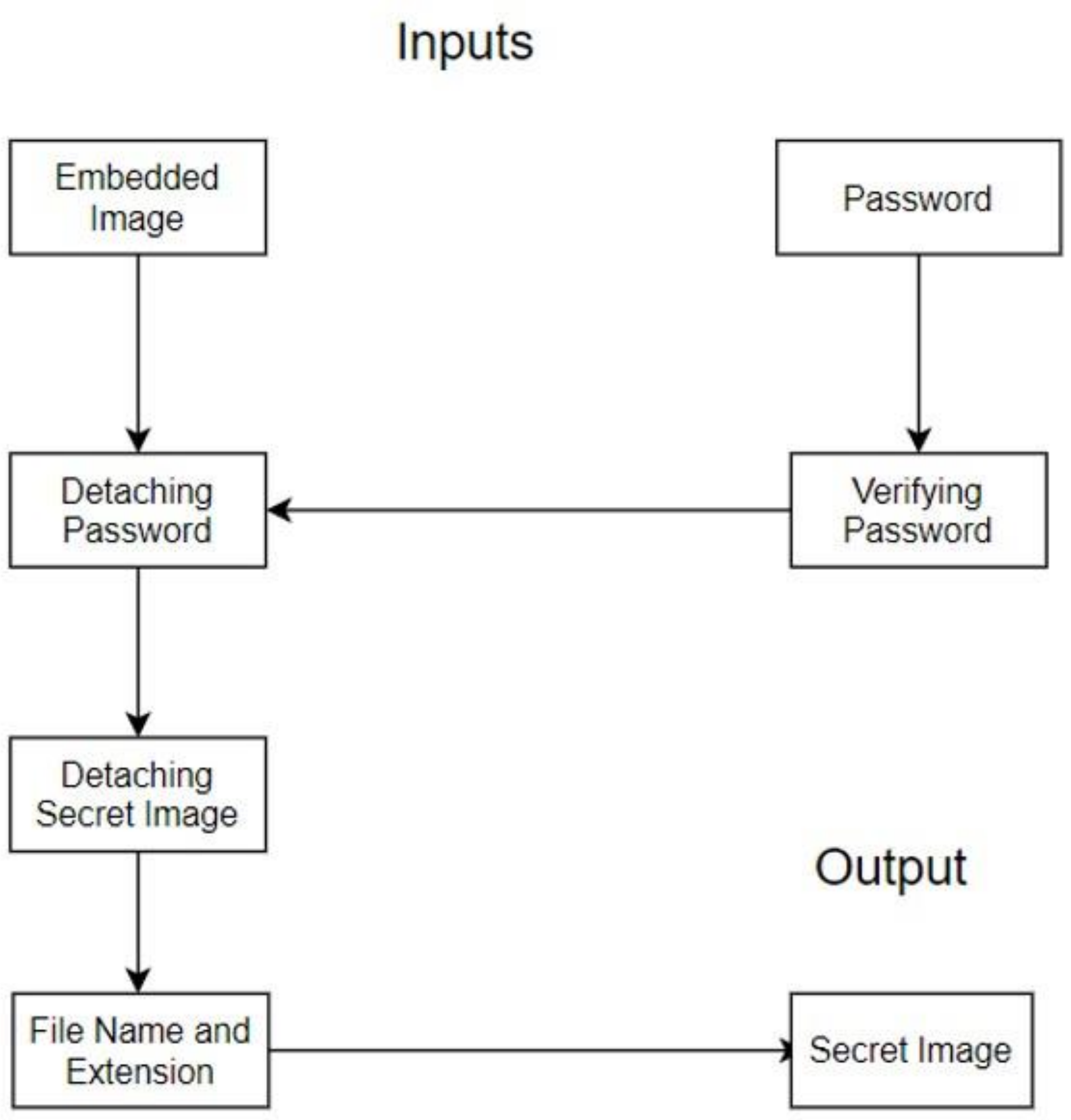
For the filename embedding feature, the program converts the file name to the ASCII numbers first, using those ASCII numbers to get the binary for the LSB insertion process.

Embedding Process



The embedding process is straight forward. The program processes all the inputs using different methods but eventually converts them into binary arrays. Then, the program combines these binary arrays together using media array as the base to form the embedded (media) image. (Note the program does resize the images to make the two images fit the best and minimize the visual impact when embedding)

Detaching Process

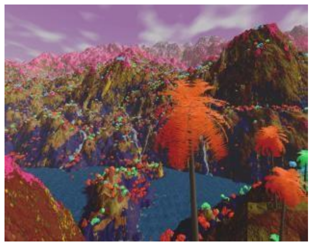


The method used to detach the secret image from the embedded (media) image is also straight forward. The program detaches the password hash from the embedded (media) image and combines them with the user entered password's hash value. If it matches, the program starts to detach the secret image and save the secret image with its original filename and extension into the output folder.

Testing

Images used for testing

Test Cases \ Images Used	Secret Image	Media Image

Happy Path Testing (Case 1 -4)	A yellow square in jpg format.	A forest in bmp format. 
Unhappy Path Testing (5 – 6)		

Happy Path VS Unhappy Path Definition:
Happy path testing:
This includes all the tests cases to simulate the normal usage of end users.

Unhappy path testing:
This includes all the test cases to test the potential vulnerabilities of the program.

Test Cases

Happy Path Cases:

#	Description	Tools Used	Expectations	Actuality	Pass/Fail
1	Encode secret image file into the media image	Stegano executable	The image file is successfully hidden inside	File is successfully hidden	Pass
2	Decode secret image file from the encoded image	Stegano executable	The image file that was hidden was successfully decoded with original name and format	File is successfully decoded with original name and format	Pass
3	Compare Images visually	Photoshop	There should be no differences when looking at it with a naked eye	There's no visible differences we can see	Pass
4	Compare Images RGB	Photoshop	The difference in RGB should be minimal between the original image and the decoded secret image	There is only minimal difference in RGB between the original image and the decoded secret image	Pass

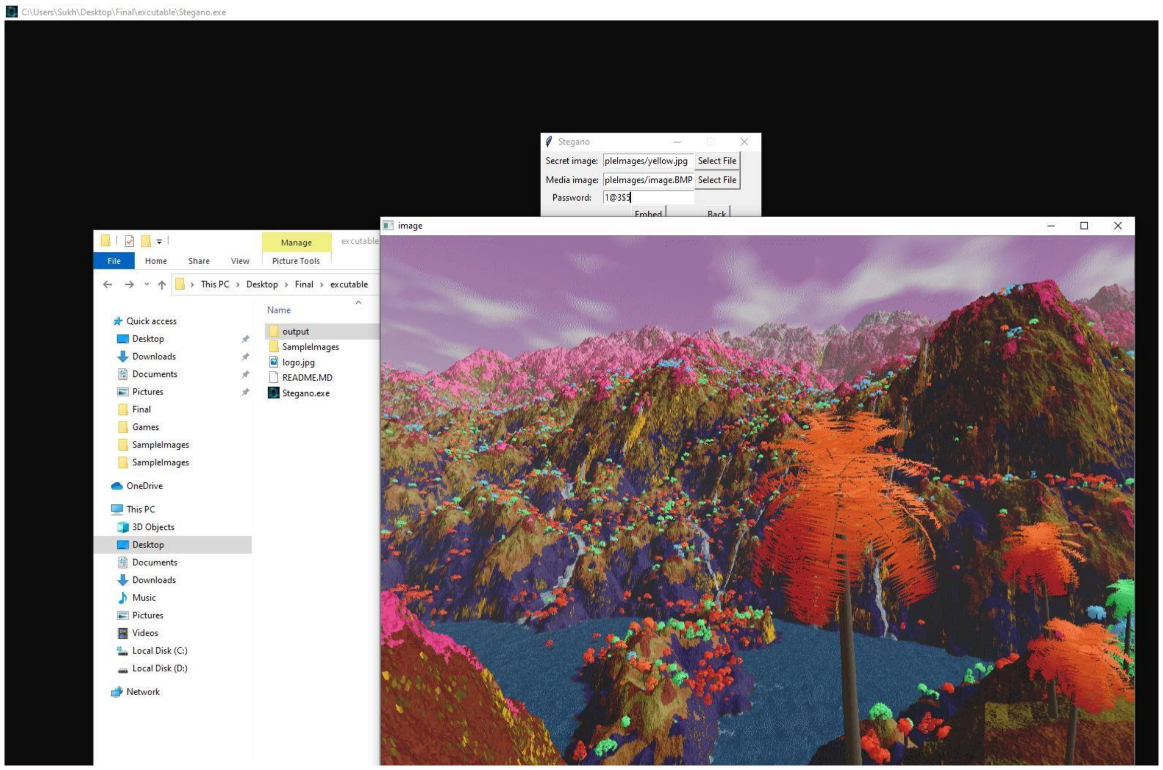
Unhappy Path Cases:

#	Description	Tools Used	Expectations	Actually	Pass/Fail

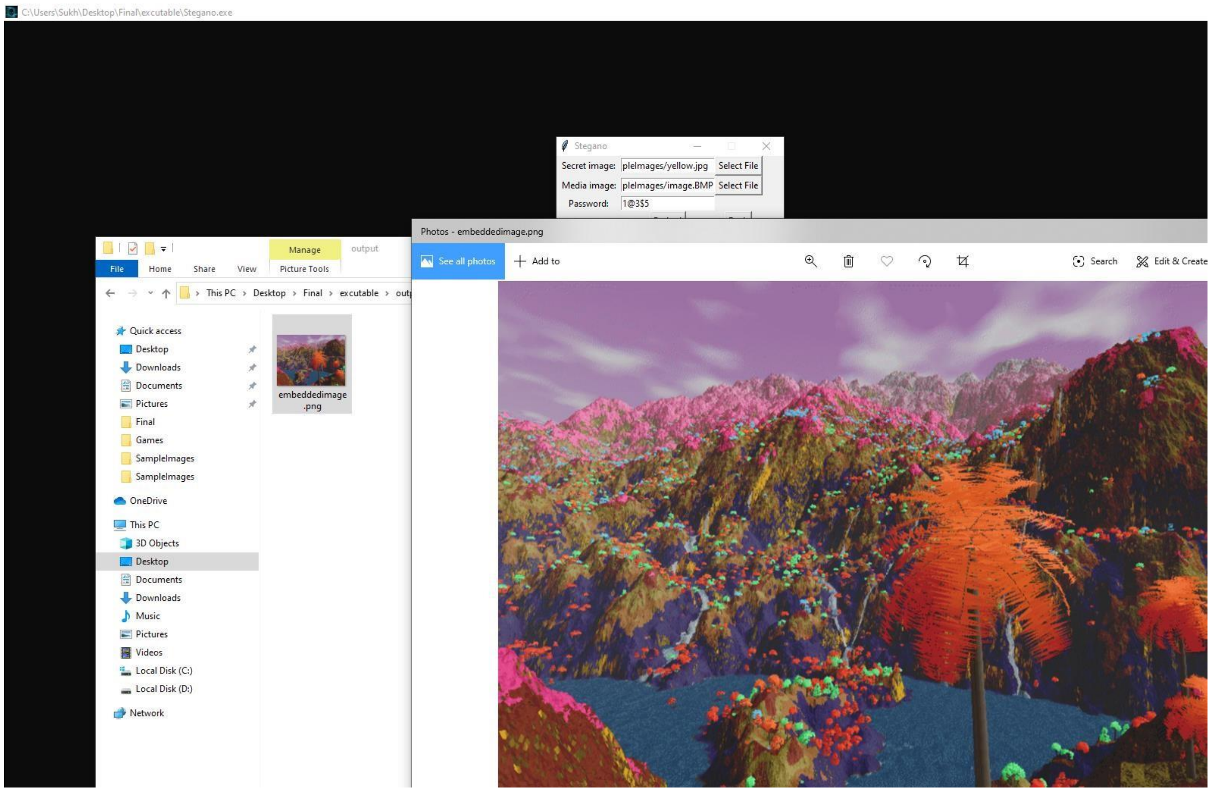
5	Hide secret image to a relatively plain image and compare the images visually	Stegano executable	There should be no differences when looking at it with a naked eye	There's no visible differences we can find	Pass
6	Try to decode the secret image by entering incorrect password	Stegano executable	The program should not start the decoding process when the entered password does not match the stored password hash	The program did not start the decoding process when the entered password is incorrect	Pass

Happy Path Testing

Test 1


C:\Users\Sukh\Desktop\Final\excutable\Stegano.exe
Stegano
Secret image: pleImages/yellow.jpg
Select File
Media image: pleImages/image.BMP
Select File
Password: 1@3$5
Embed
Back
image
This PC › Desktop › Final › excutable
output
SampleImages
logo.jpg
README.MD
Stegano.exe

Here I am using the yellow image as the secret image and the forest as the media image along with the password, 1@3$5.


C:\Users\Sukh\Desktop\Final\excutable\Stegano.exe

Stegano

Secret image: pleImages/yellow.jpg

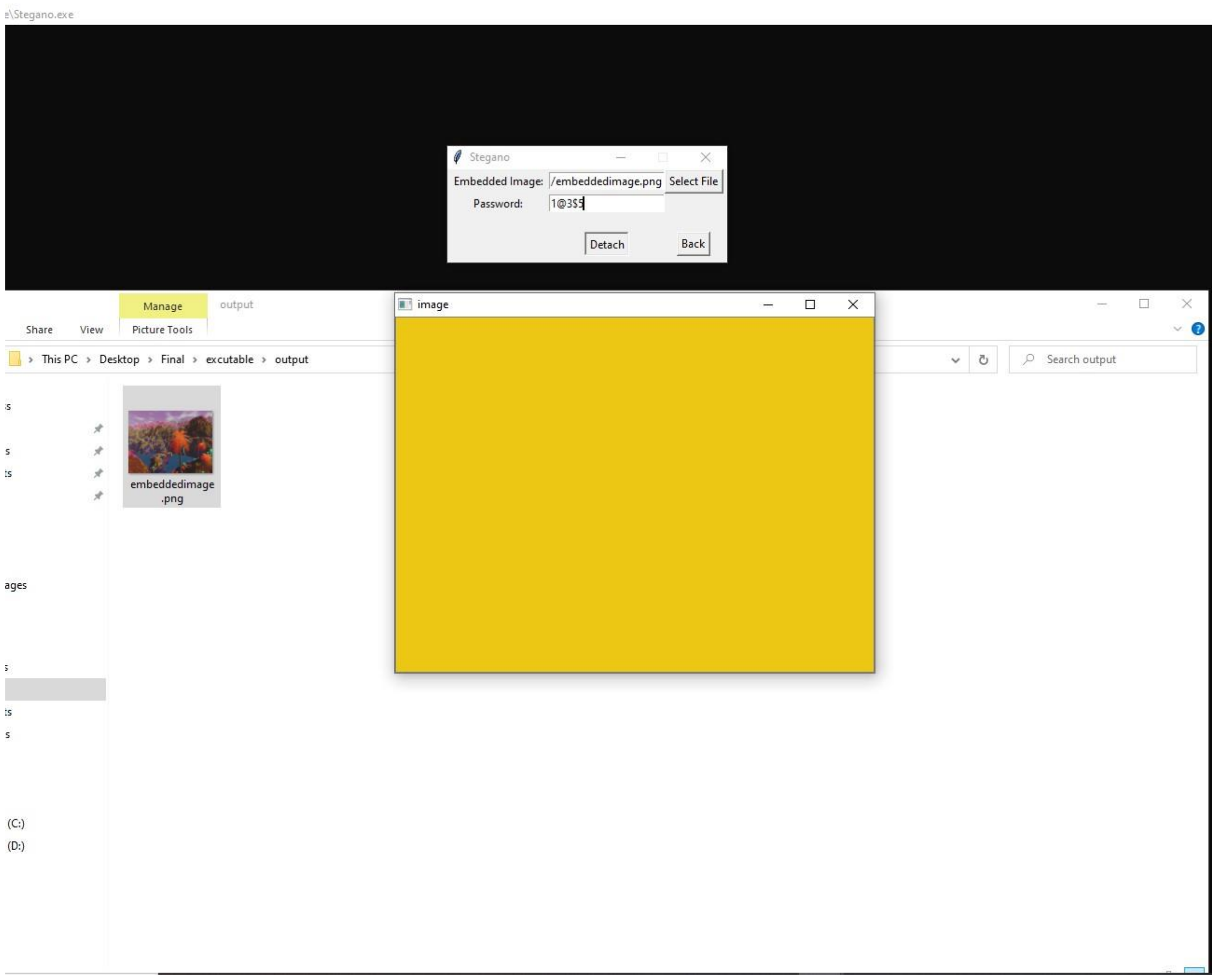
Media image: pleImages/image.BMP

Password: 1@3$5

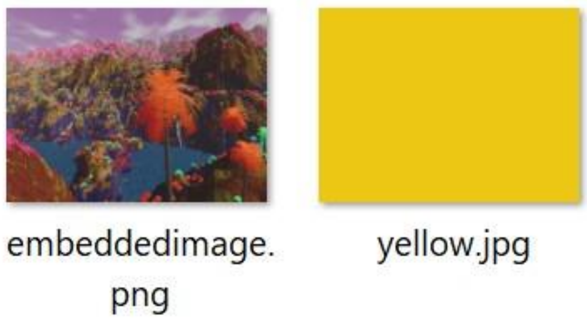
Photos - embeddedimage.png

You can view the file in the ‘output’ folder, showing that it has been successfully hidden.

Test 2

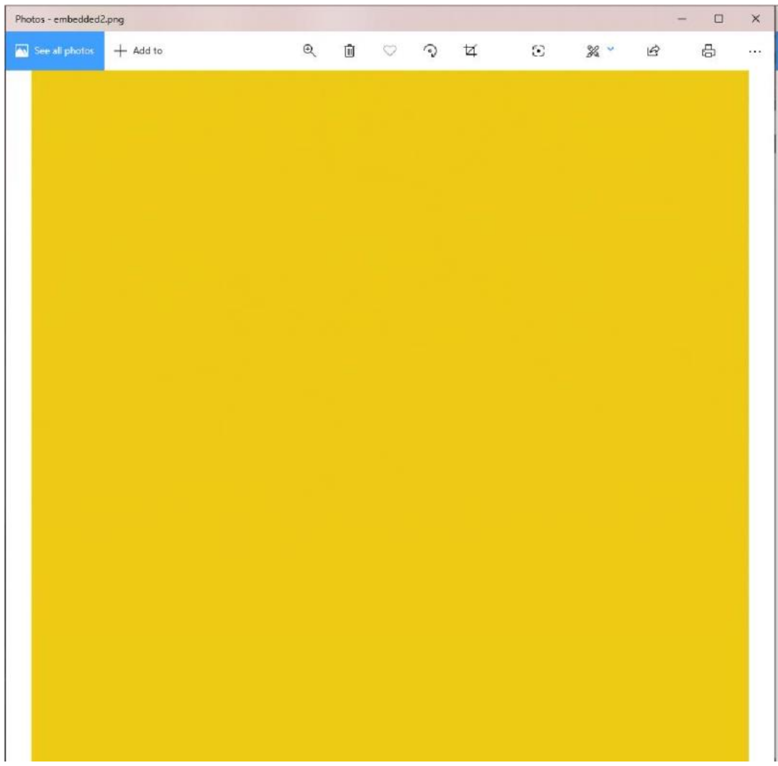


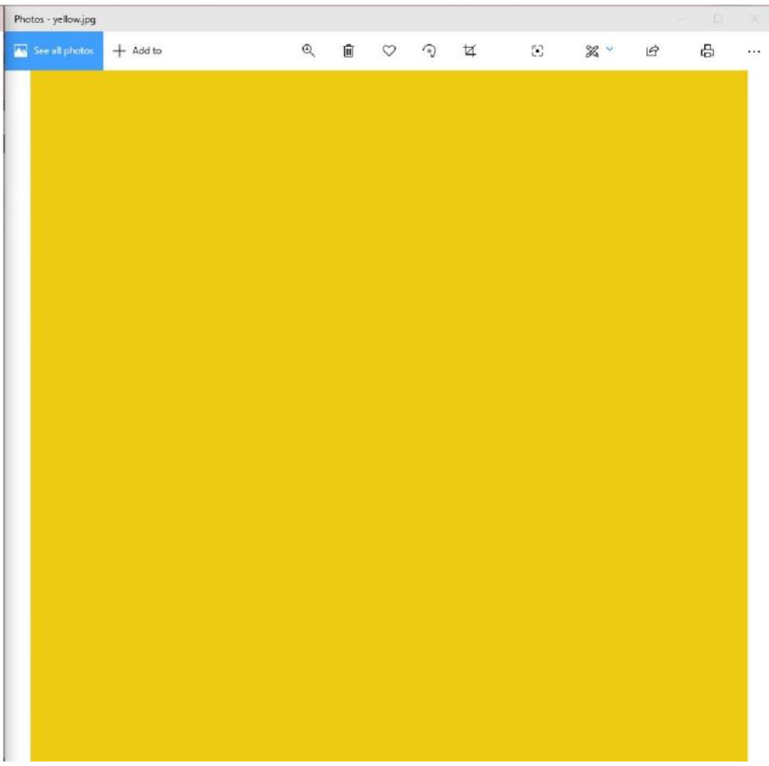
Once we choose to detach the image and putting the correct password, it showcases the image that was hidden away previously.



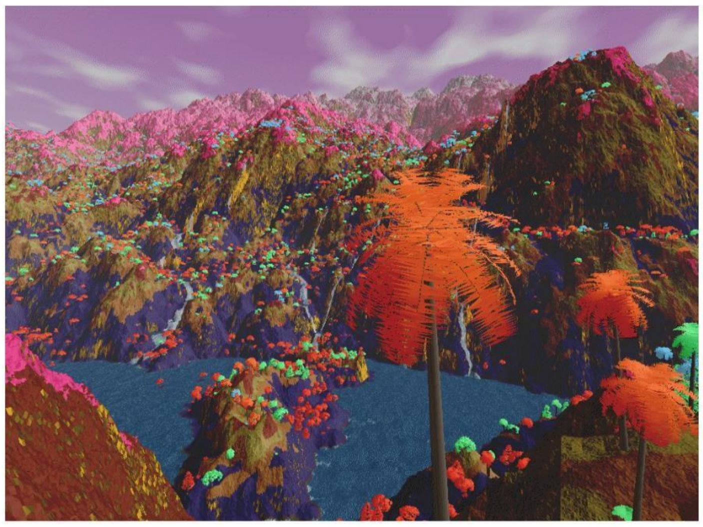
If we check the output folder, the previously hidden image is there with its original name and file format.

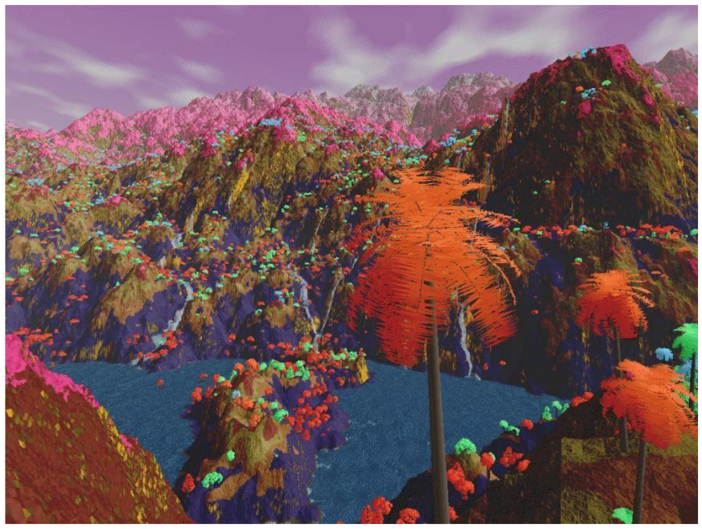
Test 3





Here we are comparing the yellow jpg file. The one on the left has been embedded with the forest, the one on the right is the original image.

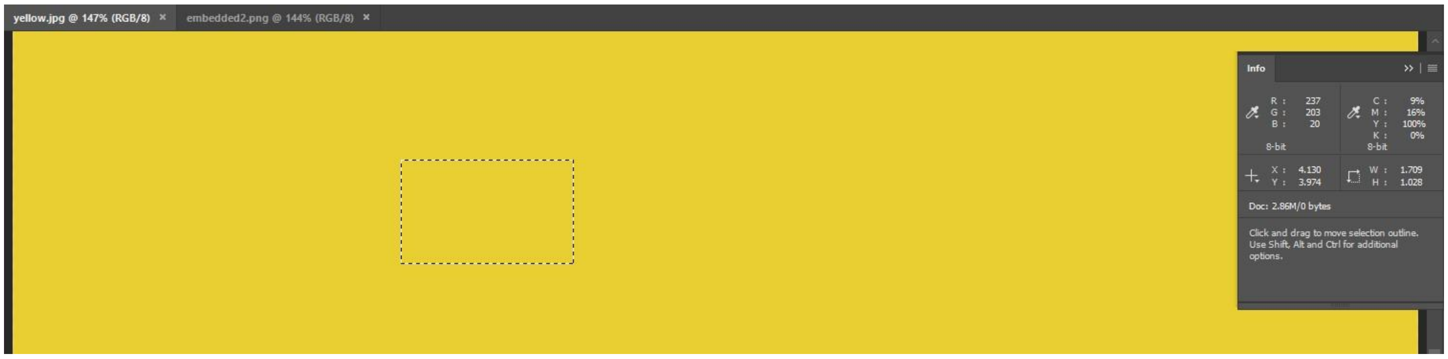




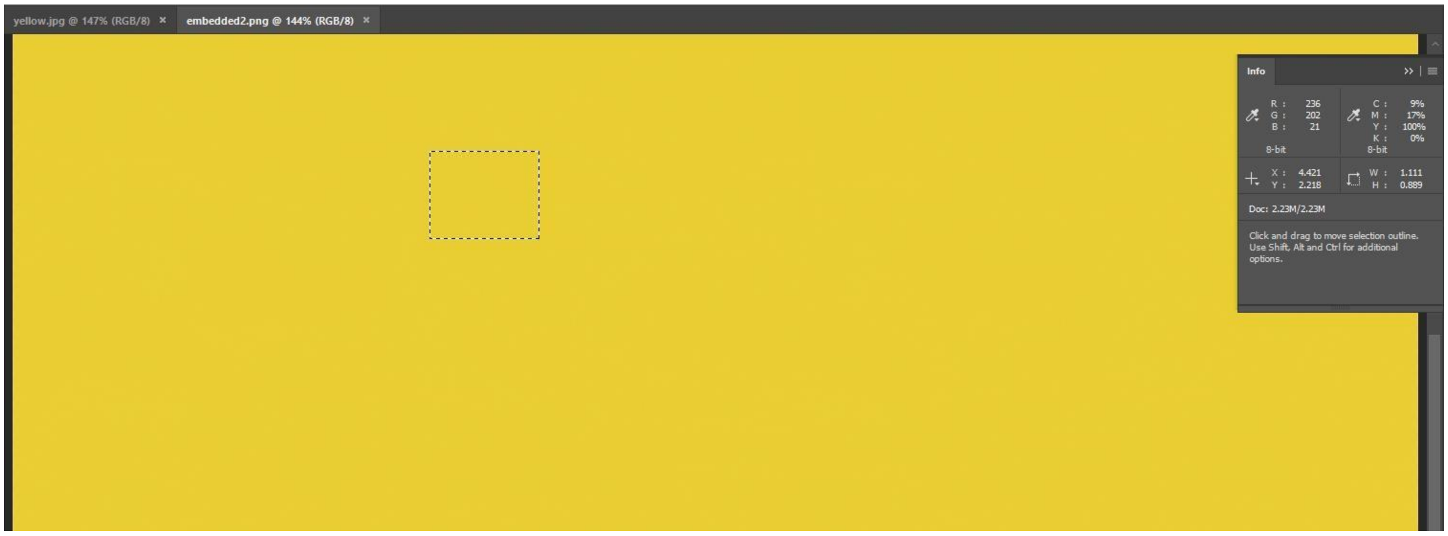
Here's the comparison for the image of the forest with the yellow jpg file embedded. Same as before, the one on the left has been embedded, the one on the right is the original.

As you can see, you can't notice any difference with the naked eye. If you saw these images on their own, you wouldn't be able to tell which one has been embedded with the secret image or if it has been embedded at all.

Test 4



Click and drag to move selection outline. Use Shift, Alt and Ctrl for additional options.



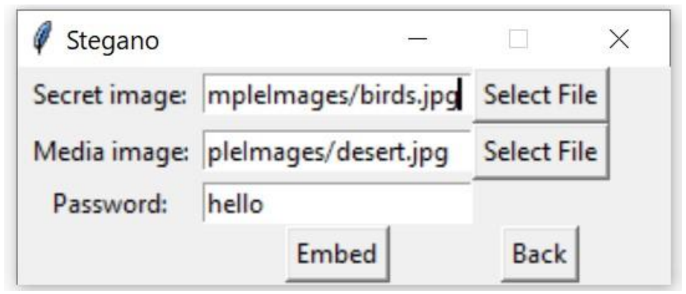
Click and drag to move selection outline. Use Shift, Alt and Ctrl for additional options.

The first image is the original, the second is the one embedded. As you can see from looking at the RGB values, there is a subtle difference between the two. The original has RGB of 237, 203, 20 throughout the entire image. Meanwhile, the embedded image has RGB of 236, 202, 21 which fluctuates depending on where you place your cursor. We've included two gif images, showcasing the change in RGB throughout the embedded image and the static RGB throughout the original image in the folder. As we can see the change in RGB is very minimal (by 1 in this case), readers of the image should not be able to find any artifacts of image manipulation.

Unhappy Path Testing

Test 5

For this test, we will embed the image on the left side (birds) into the image on the right side (desert).

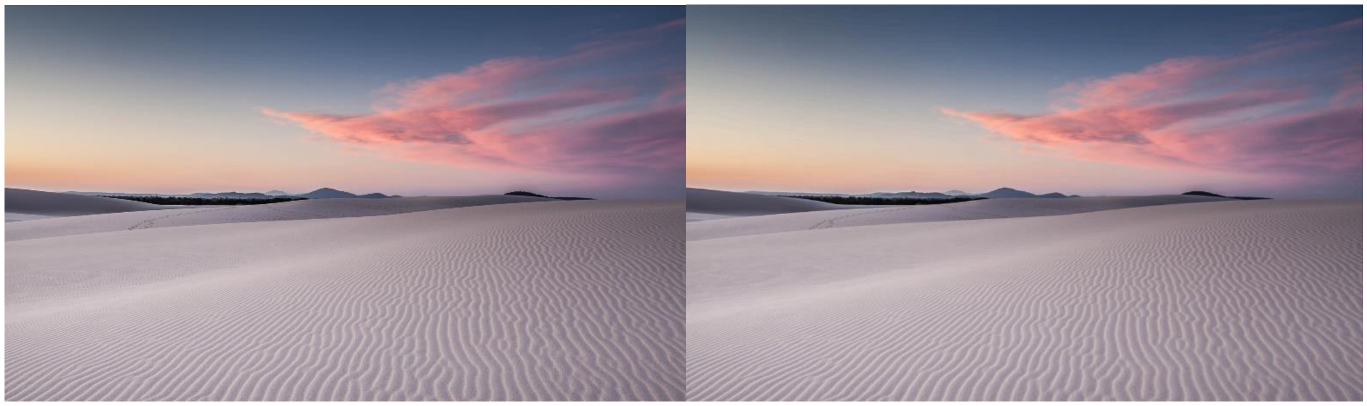


Secret image: mpleImages/birds.jpg

Media image: pleImages/desert.jpg

Password: hello

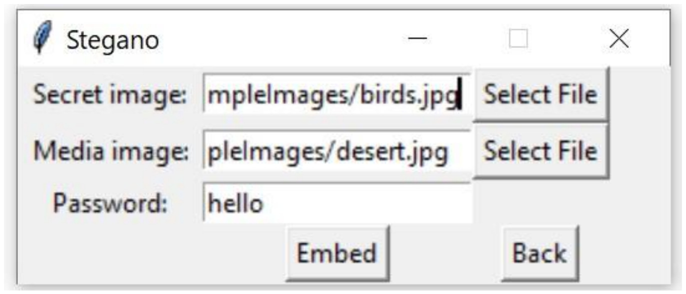
After embedding the image, now we can compare the original media image and the embedded (media) image to see if there is any visible difference.



The image at the left side is the original image, and the one at the right side is the embedded (media) image.

Comparing these two images, we are not able to find any visible differences.

Test 6



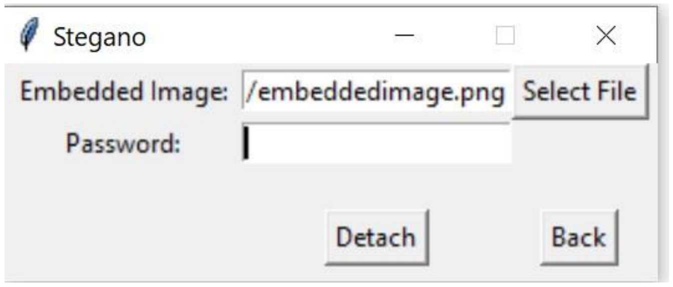
Secret image: mpleImages/birds.jpg

Media image: pleImages/desert.jpg

Password: hello

For the test, we will use the embedded image, which was created in Test 5, the password of which was set to "hello".

Firstly, we will try to see what happens when we click the "Detach" if no password is entered.

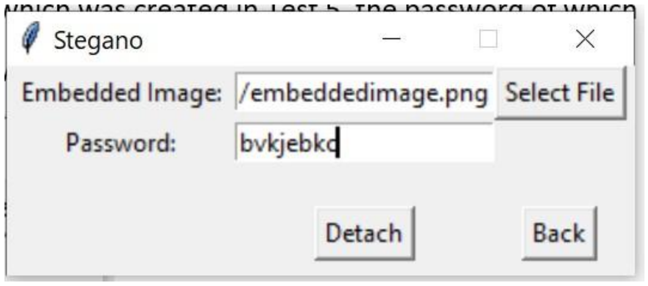


Embedded Image: /embeddedimage.png

Password:

Nothing was happened after we clicked the "Detach" button without entering any password.

Now, we will try to use a random password to see if the program will take it.

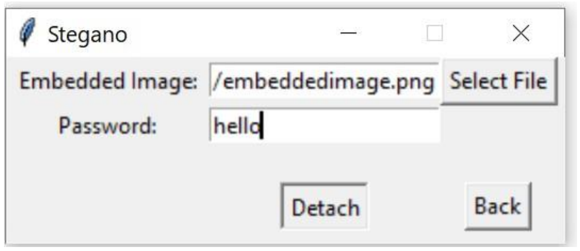

Stegano

Embedded Image: /embeddedimage.png

Password: bvkjebkd

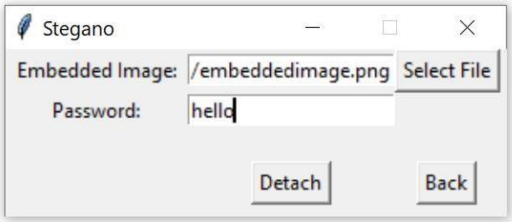
Again, nothing was happened, the program does not return any messages that indicates the password is incorrect (Which is what we want).

Finally, we will try to enter the correct password “hello” and see if the program decodes the secret image.


Stegano

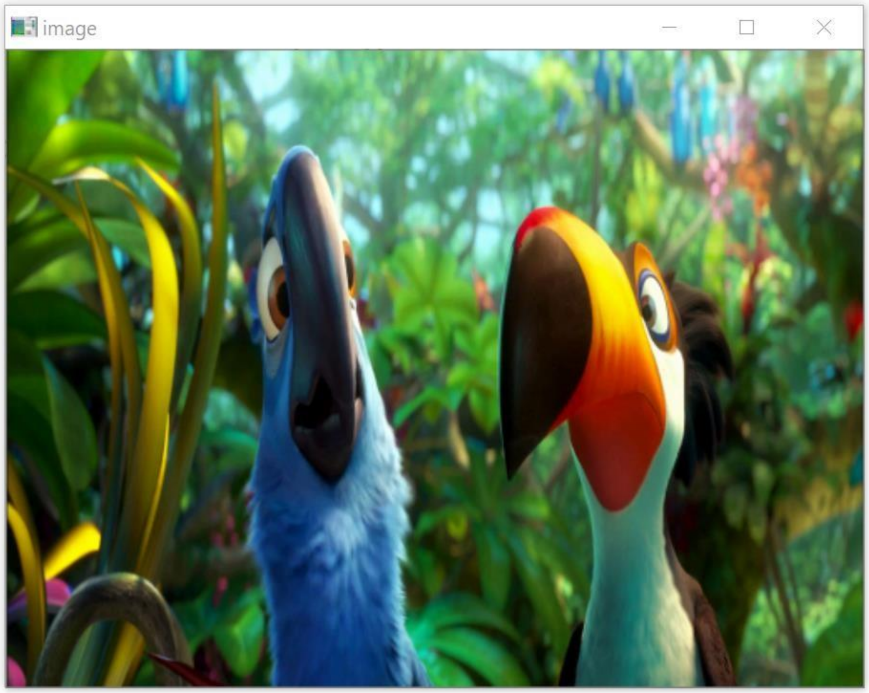
Embedded Image: /embeddedimage.png

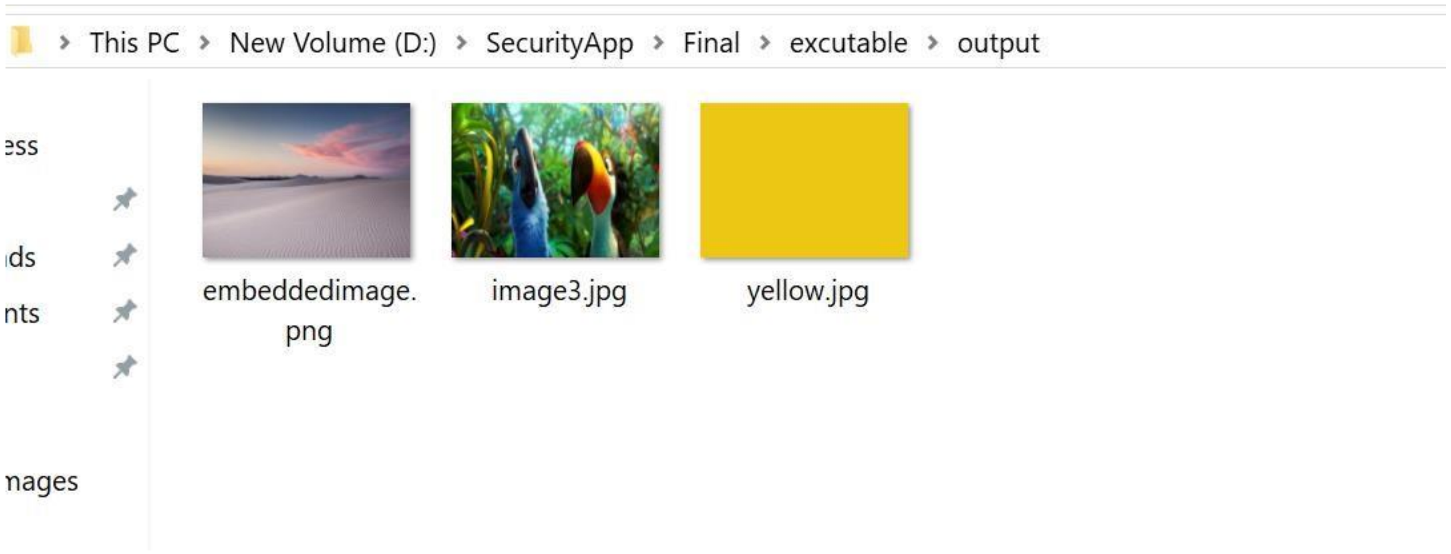
Password: hello


Stegano

Embedded Image: /embeddedimage.png

Password: hello




This PC › New Volume (D:) › SecurityApp › Final › excutable › output

As we can see, the secret image was decoded successfully and saved to the output folder.