

COMP 8505 Assignment 2

Steganography with LSB

Xinghua Wei

A00978597

Introduction

Steganography is a method to conceal and hide information in a graph. The technique involves a carrier which conceals the actual information. So, anyone other than specific receiver could not retrieve real information hidden under the carrier.

In this assignment, I will be implementing a Steganography program to hide another image within a carrier image with least significant bit insertion method.

Constraints

- The image format for the application should be BMP
- Implement encryption as part of the application
- Use LSB insertion
- Use 8 bytes in the cover image to hide a 1 byte of the secret image.
- Show testing and supporting data



Design

Platform and library

- Python 3.8.3
- PIL (pillow) for manipulating images
- Tkinter for GUI

Overall Design

Each color channel R, G, B is represented by a value, usually decimal values. And once the values are converted to binary, for example, 10 in binary is "10010". We could manipulate the least significant bit to store a binary value without changing the color too much.

For example, if R, G, B value is 120, 120, 120. Then I end up with color . By using LSB, I changed R, G, B to 121, 121, 121, then I have the color . The two color seems no difference but actually they have a bit difference of each RGB value. Therefore, if I could manipulate every pixel of an image, and the image is large enough to hold enough information, I could insert a binary represented image to another image without noticeable color change.

About the size of an image, usually an 8-bit resolution image need $640 \times 480 \times 1$, which is 307200 bits (307KBytes) space to store. If an image is 24-bit and 640×480 resolution, it requires $640 \times 480 \times 3 \times 8$, which is 7372800 bits (921KBytes) to store.

For cipher, I will swap binary format RGB values of the secret image and swap back when decoding it. For example, if the binary represent RGB value is "11110000", I will change it to "00001111" when concealing it to carrier image. When reconstructing secret image, the "00001111" will be swap back to "11110000". Therefore, even some third part manage to extract the binary value of secret image. The value would not make any sense. Besides, I will use Ceaser to encrypt password and add encryption to output filename. The receiver need correct password to decode the image.

Functions

Gui.py

Gui.py is the file for GUI. It involves tkinter, PIL, subprocess libraries. This provides convenience to users and security ability to hide process under it.

- Encode_gui():
 - Create a encode image canvas with password input field, output image name field, carrier image select field, carrier image type (bmp or jpg) select field, secret image select field, encode button to check and start encode process, back button to return to the main canvas.

Password(No number)

File Output Name

Open Carrier File

☐ bmp

☐ jpg

Open Hide File

ENCODE Back

- Inner class encode_gui():
 - Grab and validate user input
 - Call encode function from encode_image.py file
- Decode_gui()
 - Create a decode image canvas with password input field, steganography image select field, decode button to check and start decode process, back button to return to the main canvas.



- Inner class `decode_img()`:
 - Grab and validate user input
 - Call decode function from `encode_image.py` file

Help.py

`Add_zero(argument1, argument2):`

Add zero to a binary number (`argument1`), so that the number of characters in the binary string can be extended to expected length (`argument2`). Also, the value of the binary string will not change.

It returns a binary string with a length specified by the `argument2`.

`Rgb_to_binary(r, g, b):`

This function converts decimal values of RGB of a pixel into binary format of same value.

It returns binary format of same RGB value with length specific to 8 bits.

`Ceaser_enc(argument):`

This function encrypts user input password with Ceaser. Letter shift value set to 5.

It returns the encrypted message.

`Decode_cipher(argument):`

This function decrypts user input password with Ceaser. Letter shift value set to 5.

It returns the decrypted message.

`Swap_cipher(string):`

This function takes a string and swap first and second halves of the string to encrypt or decrypt the string.

It returns the encrypted string or decrypted string.

[Encode_image.py](#)

`Get_pixel_binary_value(image, width, height):`

This function scans every pixel of the selected image column by column. Then concatenate binary format of each pixel value together to a new string. Each RGB value is specified to 8 bits long.

It returns a string with concatenated binary format RGB values of each pixels in the image.

`Modify_binary_value(carrier_image, carrier_image_width, carrier_image_height, secret_image_width, secret_image_height, secret_image_binary_pixel_values):`

This function use LSB insertion to manipulate carrier image binary RGB values. It replaces 1 least significant bits of a set of pixel values in the image. This means I will use 8 Bytes to conceal 1-byte information. The first 24 bits, the first pixel, will be used to store secret image width and height. Even the secret image might be too large, it will still hide part of the image.

It returns RGB values of carrier image where the least significant bits of each set are replaced with secret image binary bits.

`Encode(carrier_image, secret_image):`

This function retrieves the carrier image and secret image and call above functions to conceal secret image to the carrier image.

It returns the modified carrier image, which is not very different from original carrier image, but secret image binary bits are concealed in the modified carrier image.

[Decode_image.py](#)

`Extract_hidden_pixel(image, carrier_width, carrier_height, counter):`

This function scans and extracts a set of bits of secret image from the carrier image.

It returns a binary format of pixel values of the secret image.

`Reconstruct_image(image_pixels, secret_image_width, secret_image_height):`

This function reconstructs the secret image using extract binary pixel values.

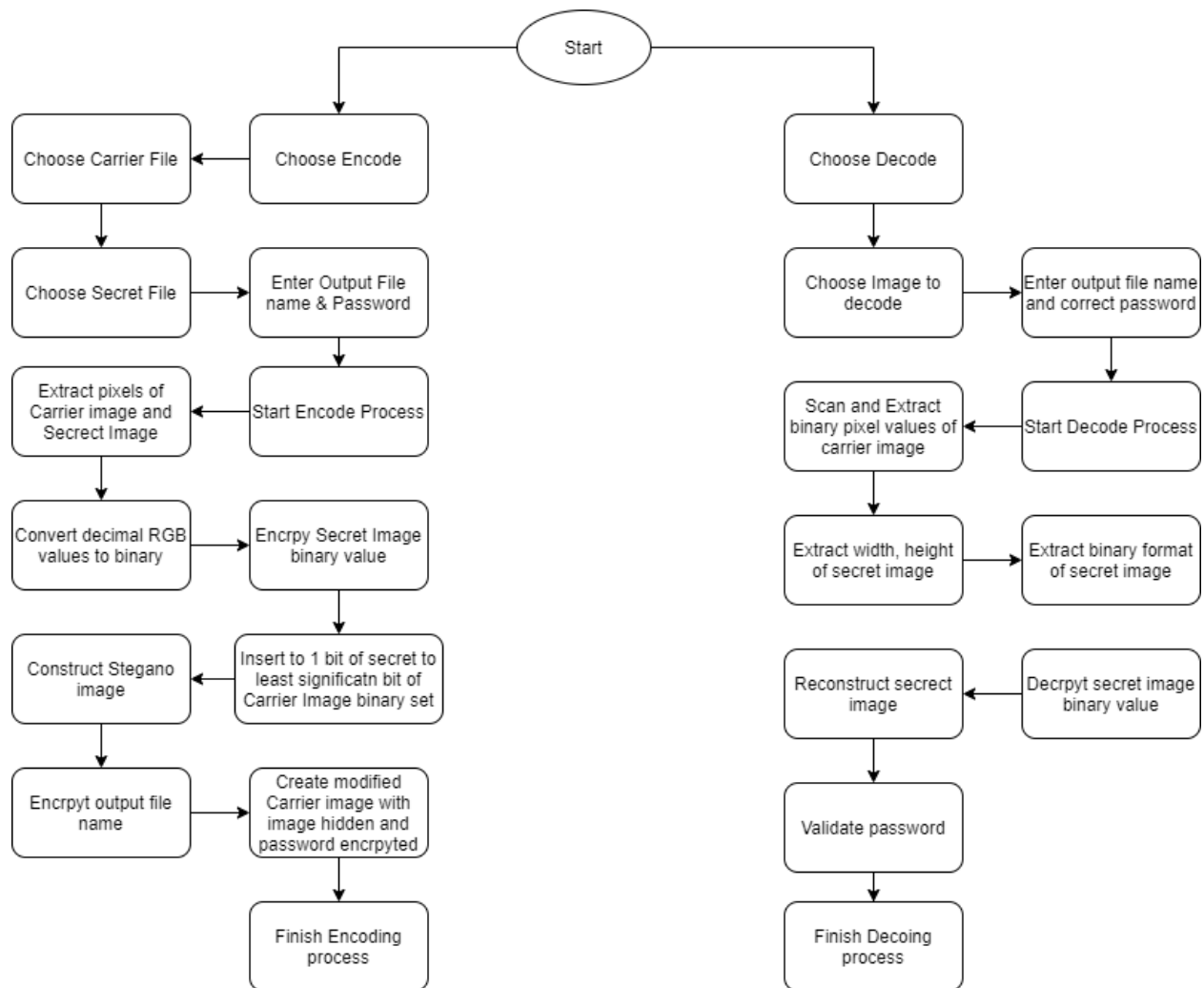
IT returns the reconstructed secret image.

`Decode_image(image):`

This function loads the modified carrier image and call above functions to construct the secret image.

It returns the secret image.

Diagram

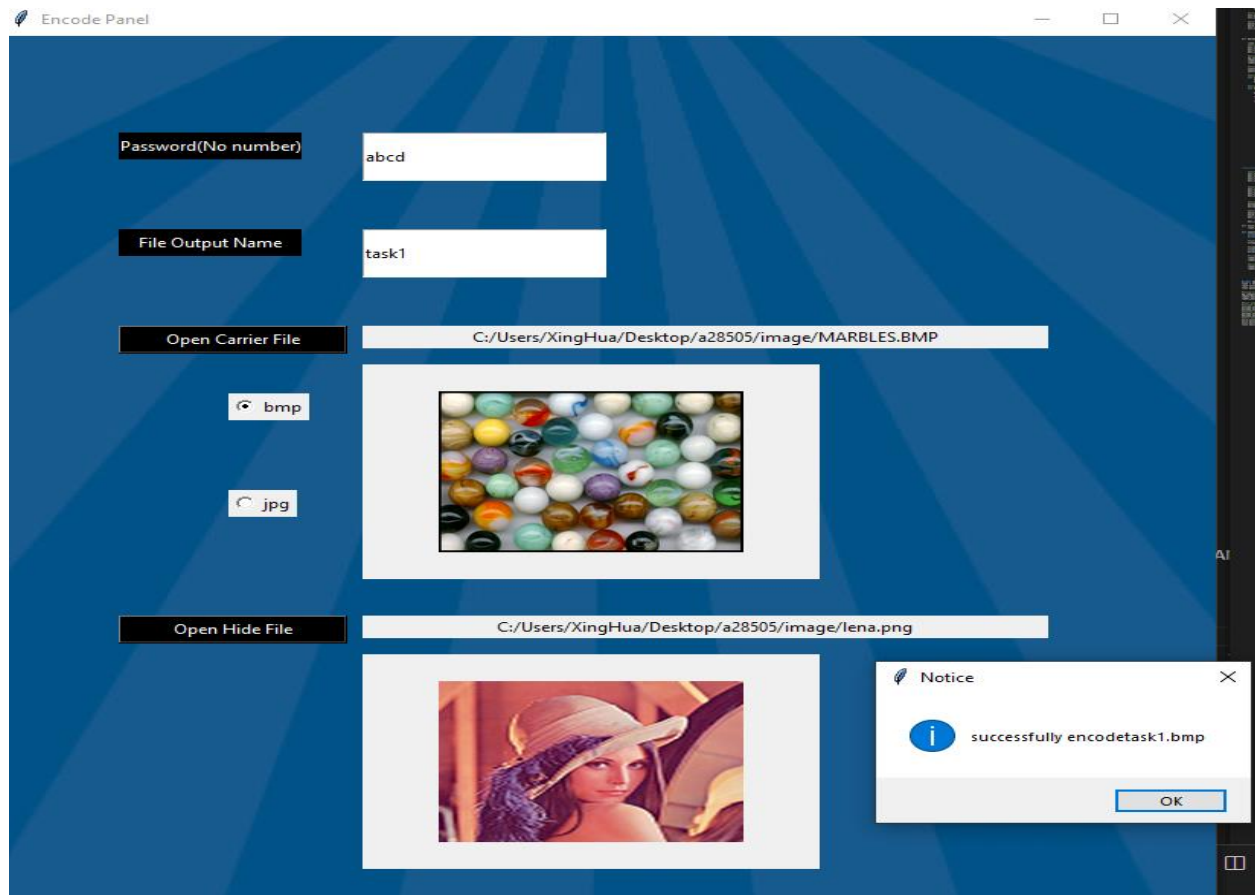


Testing & Support Data

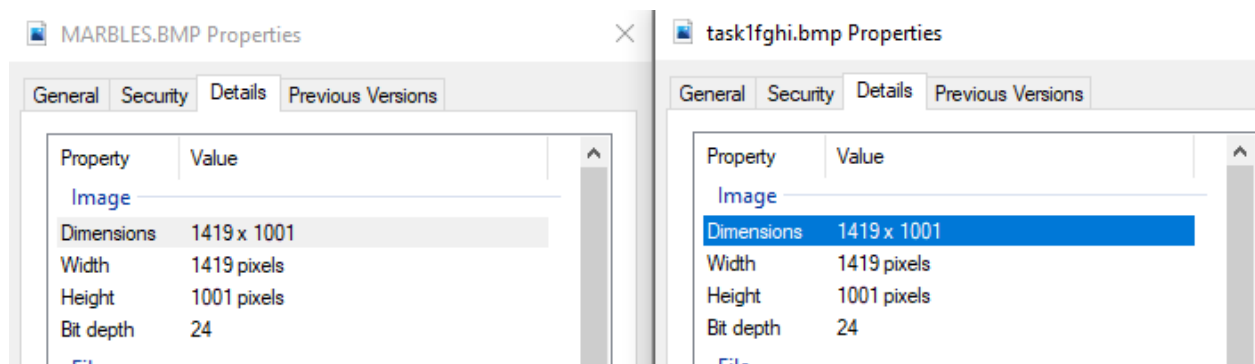
Task 1 – Hide PNG image to BMP image (BMP image with lots of color):

- BMP file: 24-bit / 4M size
- PNG file: 32-bit/ 477 Kbyte size (Requires 103*103*4 bytes size to store, may lost some pixels)

Encode:

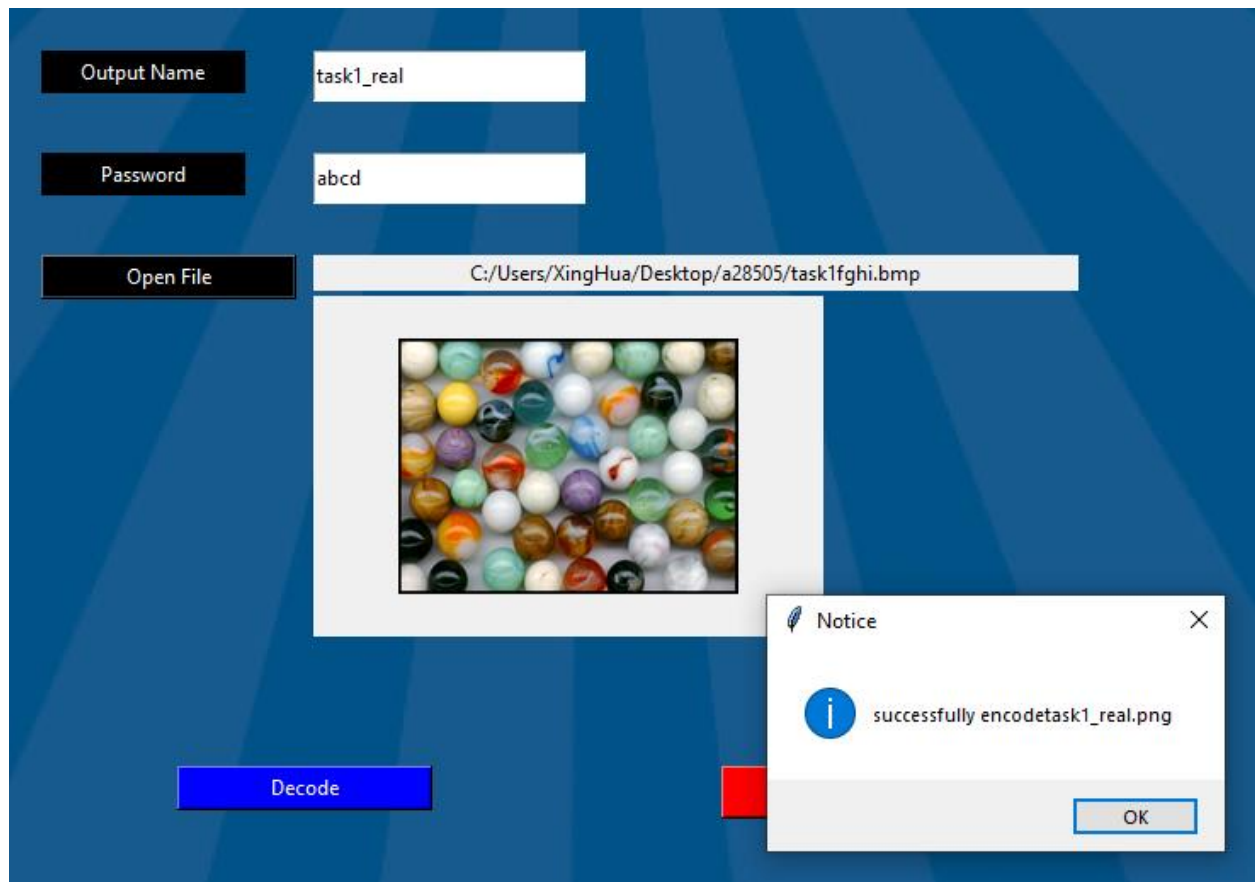


- Successfully hide secret image to carrier image

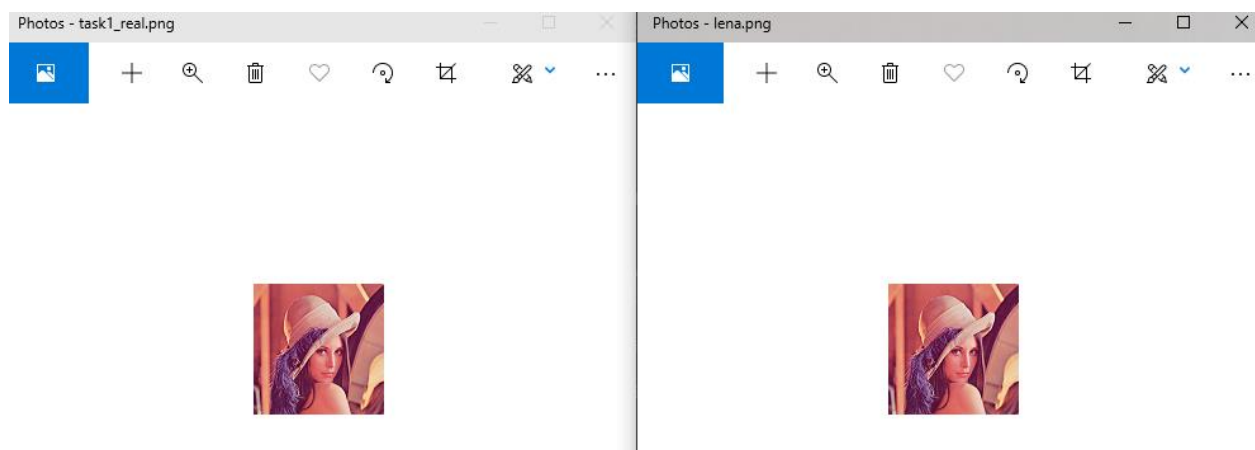


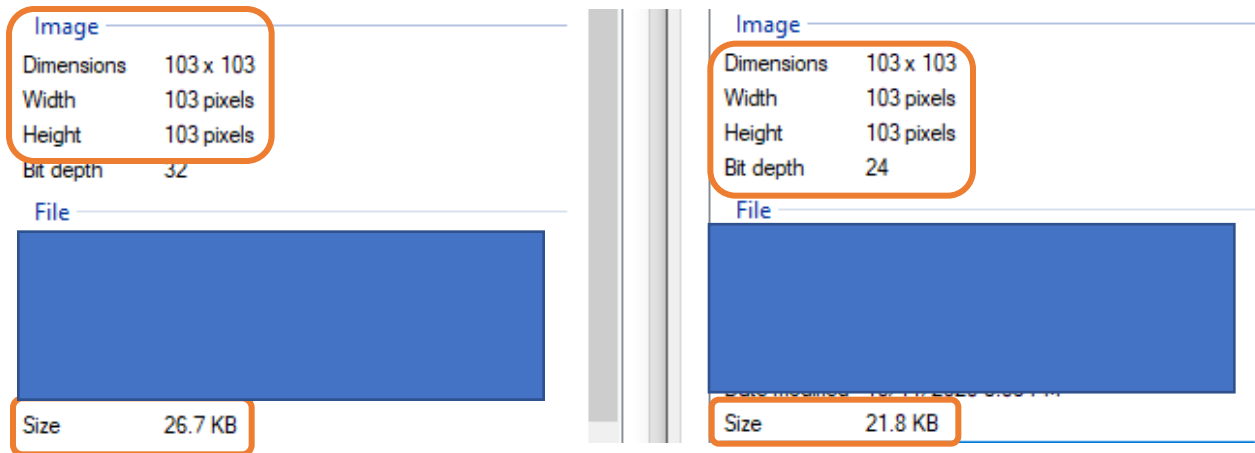
After encoding, the encoded image has nothing different from the original image.

Decode:



- Decode success



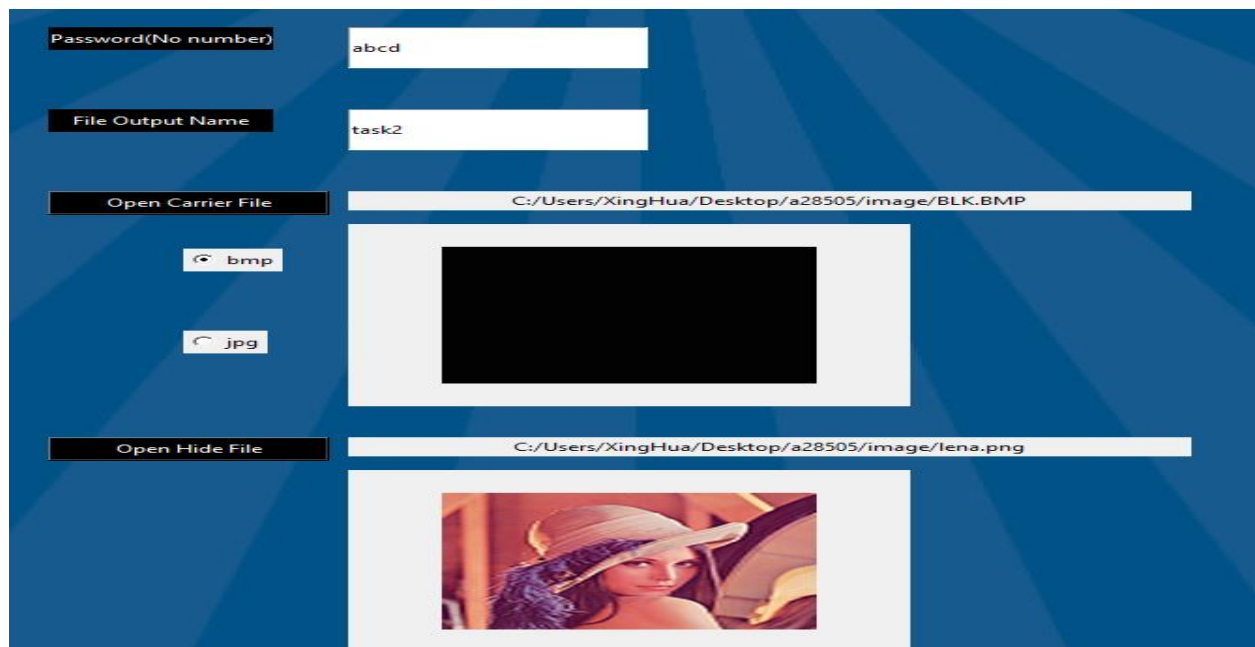


After hide and tract secret image from carrier image, there are no obvious difference when looking at them. But there is a size reduction. Overall, hide PNG image to a colorfully BMP image succeed.

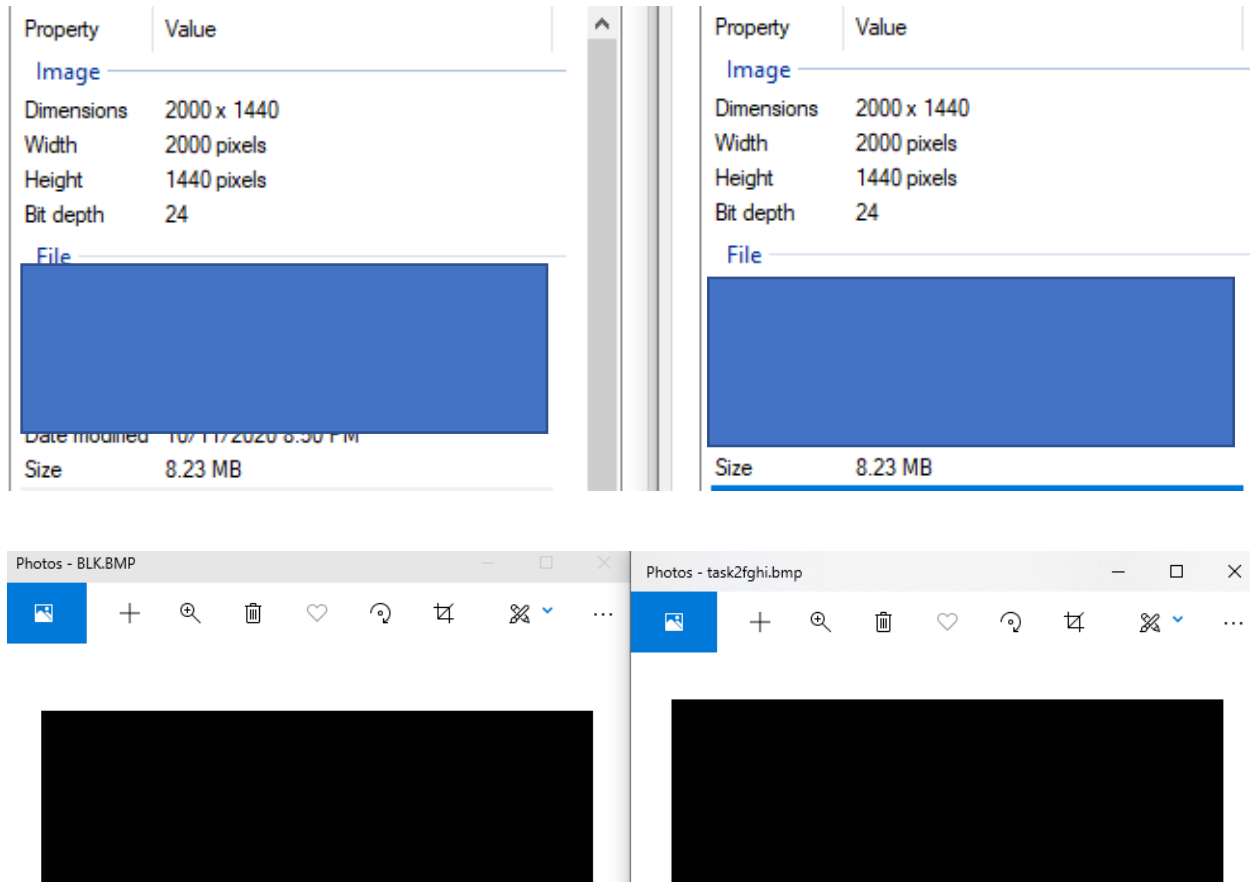
Task 2 – Hide PNG image to BMP image (BMP image with solid color):

- BMP file: 24-bit / 8M size
- PNG file: 32-bit/ 477 Kbyte size (Requires $103 \times 103 \times 4$ bytes size to store, may lost some pixels)

Encode:

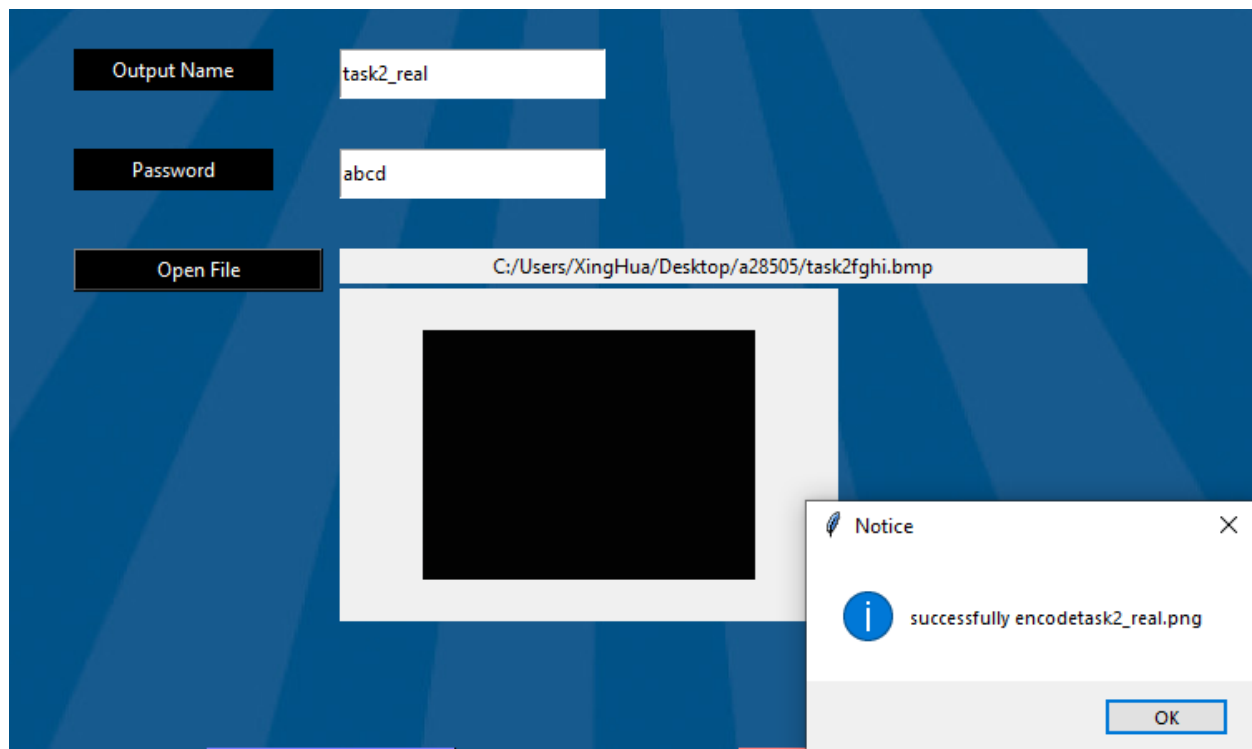


- Successfully hide secret image to carrier image

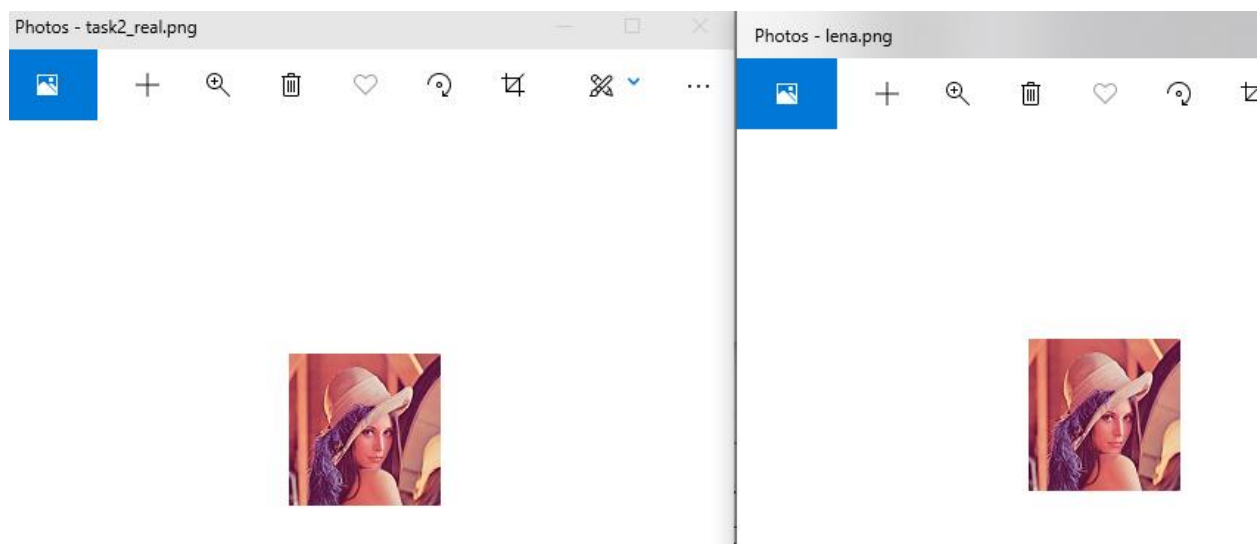


After encoding, the encoded image has nothing different from the original image. Even BMP image is a large solid color won't show any difference.

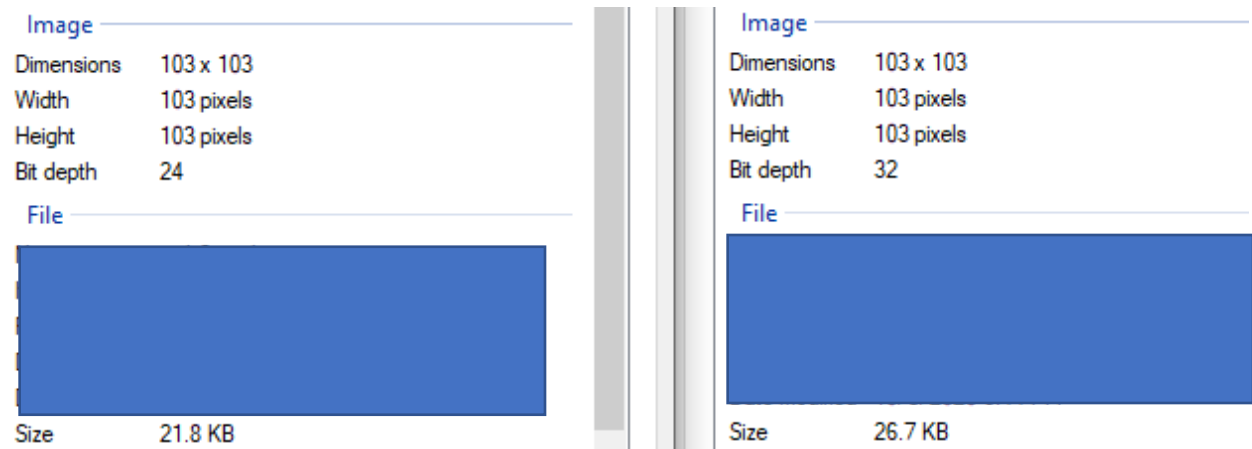
Decode:



- Decode success



Comparing two images with human eyes won't notice any difference.

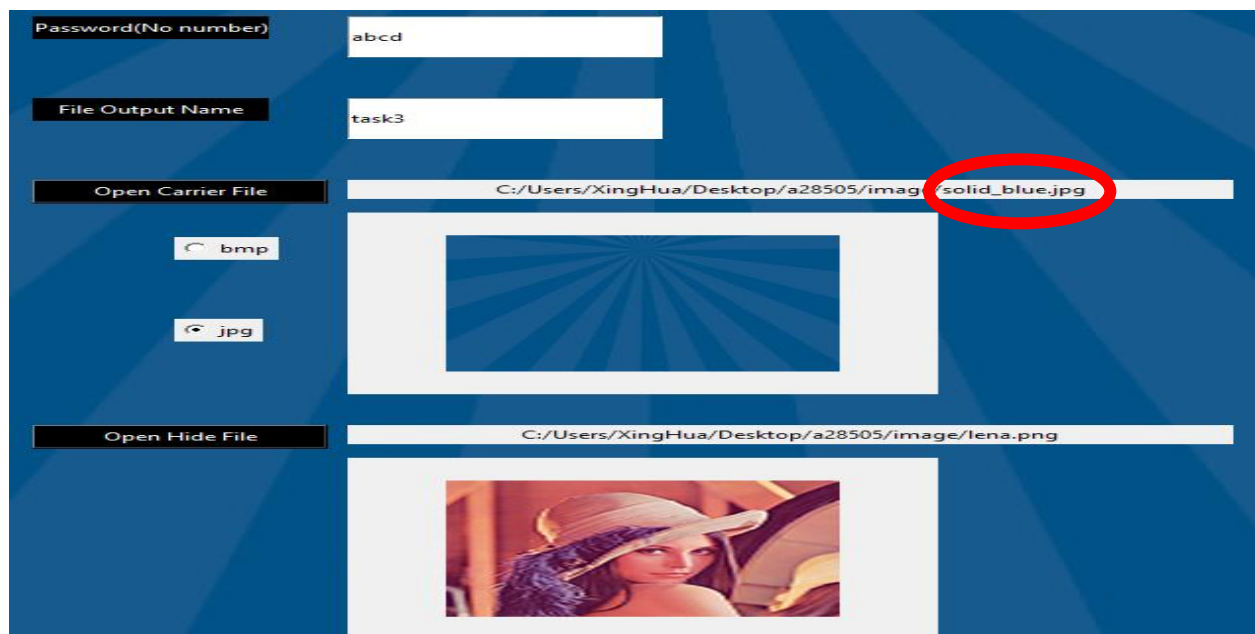


After hide and tract secret image from carrier image, there are no obvious difference when looking at them. But there is a size reduction. Overall, hide PNG image to a solid color BMP image succeed.

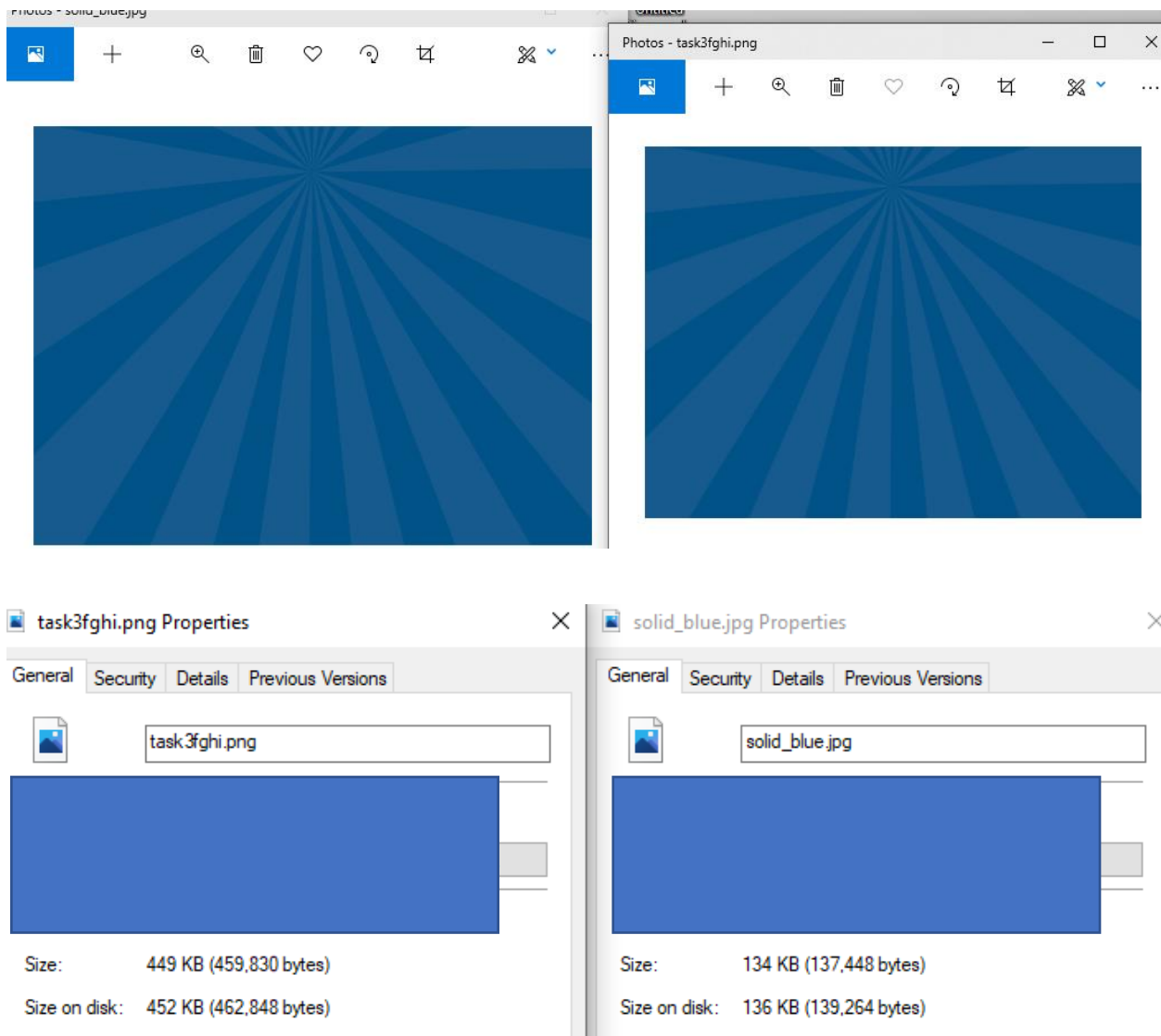
Task 3 – Hide PNG image to JPG image:

- JPG file: 24-bit / 134KB size
- PNG file: 32-bit/ 477 Kbyte size (Requires $103 \times 103 \times 4$ bytes size to store, may lost some pixels)

Encode:

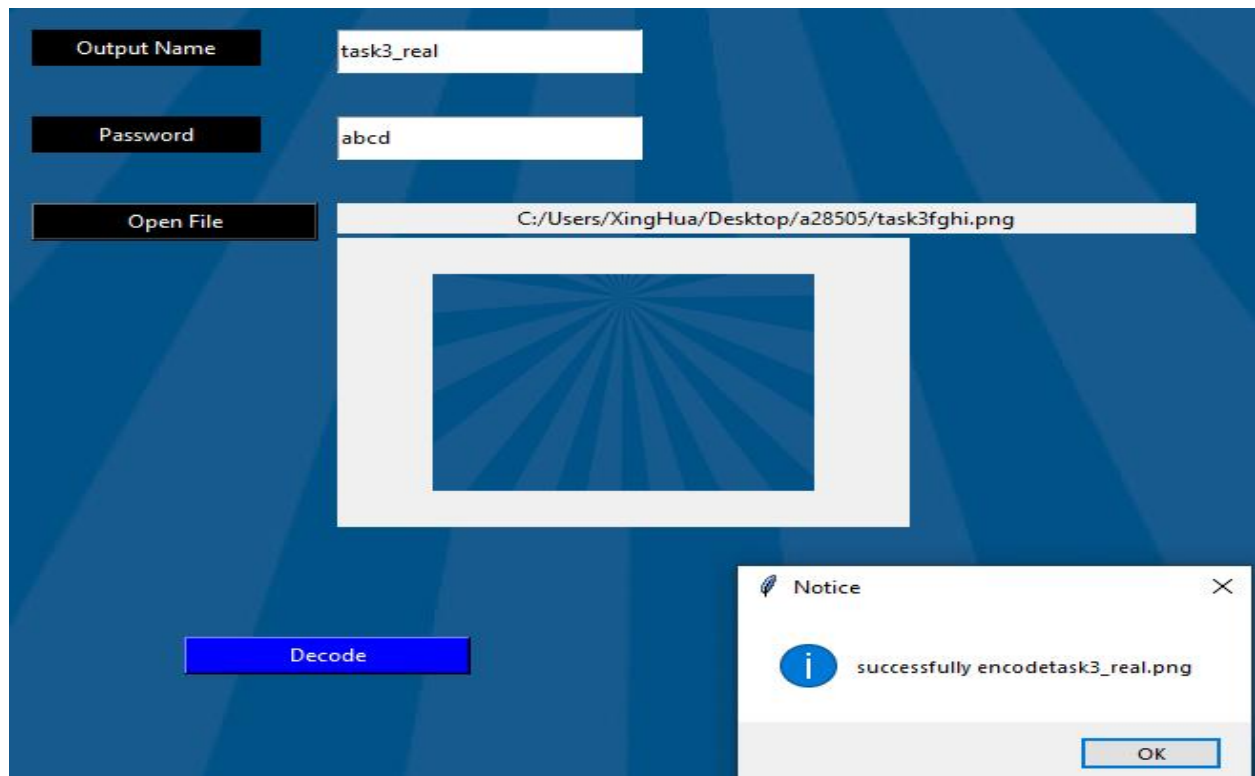


- Successfully hide secret image to carrier image

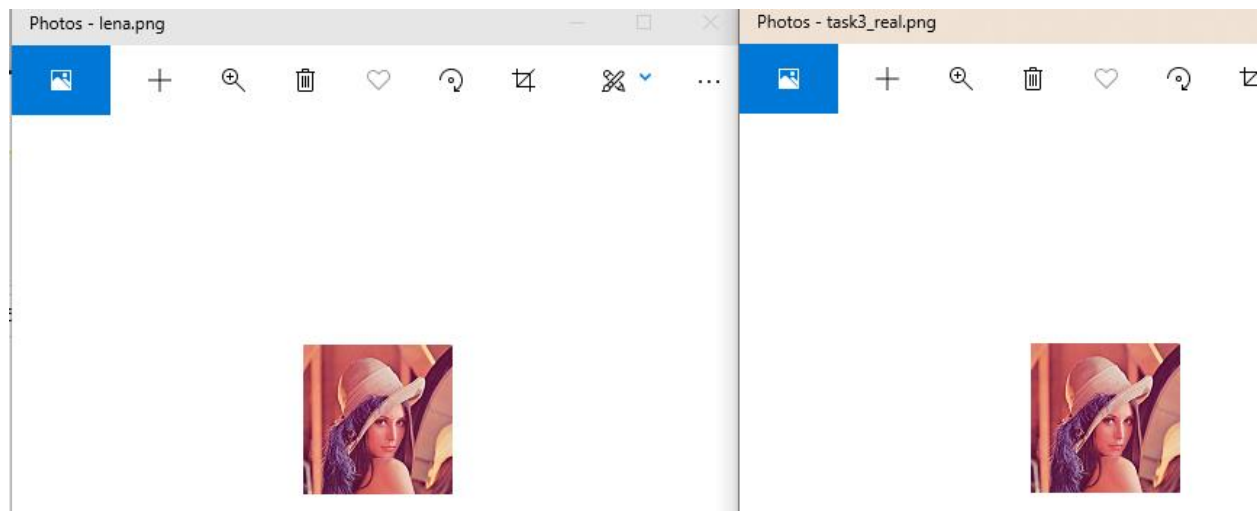


After encoding, the encoded image has nothing different from the original image. Even though the modified carrier image becomes larger. Still visually no difference.

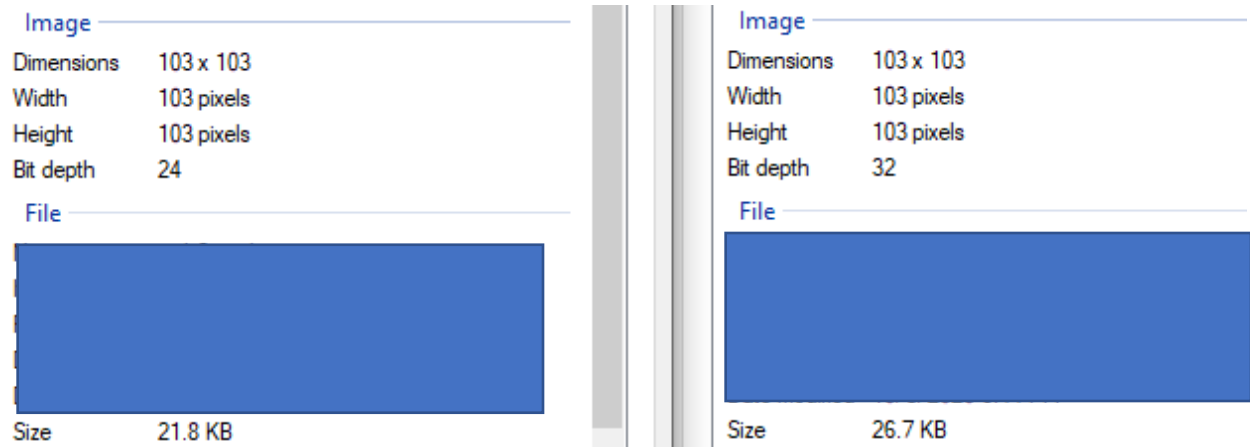
Decode:



- Decode success



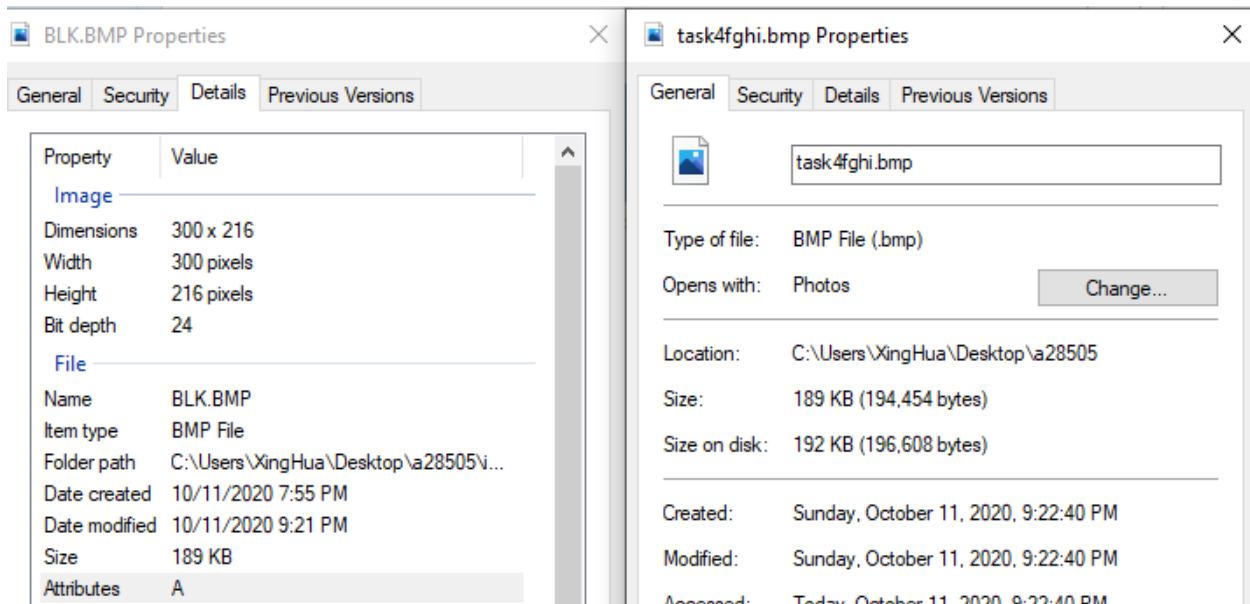
Comparing two images with human eyes won't notice any difference.



After hide and tract secret image from carrier image, there are no obvious difference when looking at them. But there is a size reduction. Overall, hide PNG image to a JPG image succeed.

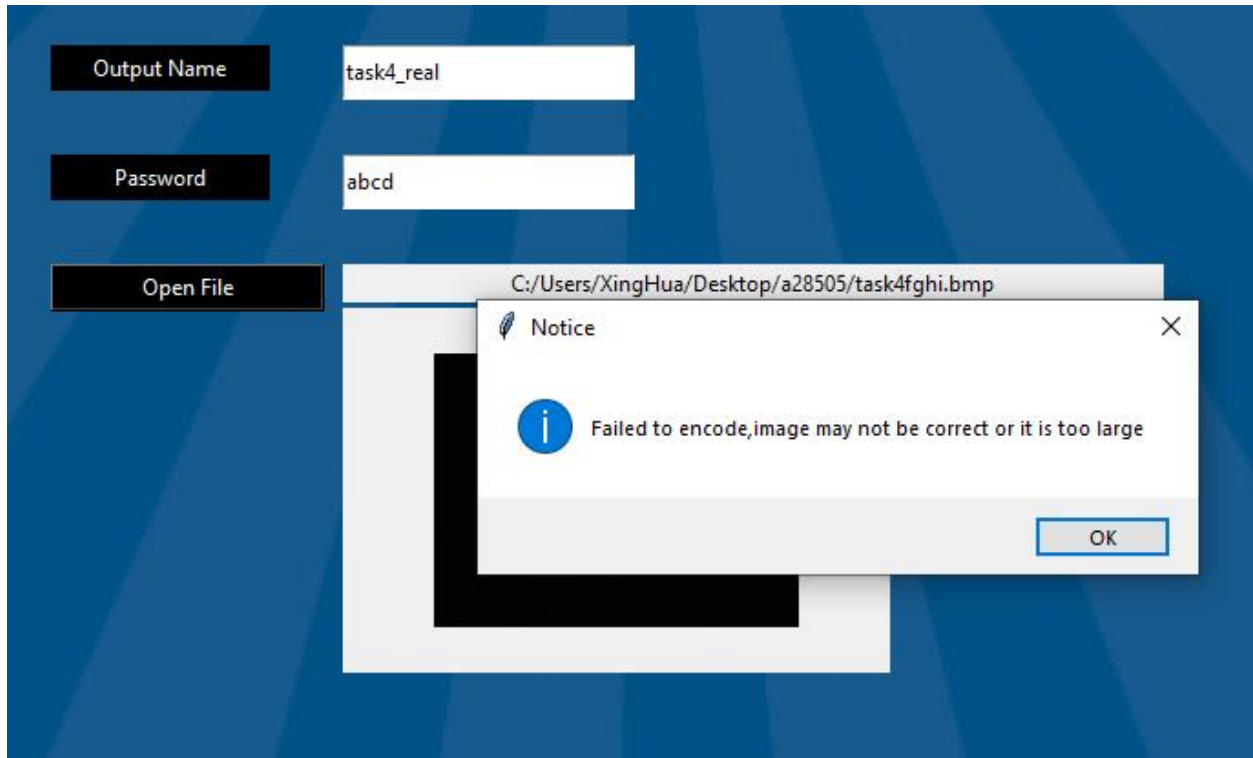
Task 4 – Carrier Image size not enough (Should fail to decode)

- JPG file: 24-bit / 189KB size
- PNG file: 32-bit/ 477 Kbyte size (Requires $103 \times 103 \times 4$ bytes size to store, may lost some pixels)



Here I created a modified carrier image task4fghi.bmp. Which theoretically should not be decoded because binary information in that image is not enough to reconstruct another image. Because my secret image needs about 4.2Mbytes space.

Decode:



When start decoding, it pop ups a notice say secret image too large. It means decode failed. Therefore, after inserting secret image to a carrier image without enough space, it will fail. Test 4 success.