IMAGE STEGANOGRAPHY

**APPLICATION DEVELOPMENT – iii**

**IMPLEMENTATION** **PHASE**

*Submitted in partial fulfilment for the award of the degree*

*of*

**Master of Technology**

***in***

**Information Technology**

*by*

**Bhuvana Chandra Pathi**

**(17MIN0618)**



**School of Information Technology and Engineering**

September 2020



**School of Information Technology and Engineering**

**DECLARATION BY THE CANDIDATE**

I hereby declare that the thesis entitled **“IMAGE STEGANOGRAPHY”** submitted by **Bhuvana Chandra (17MIN0618)** to Vellore Institute of Technology, Vellore, in partial fulfillment of the requirement for the award of the degree of **Master of Technology** in **Information Technology** is a record of bona fide APD work carried out by me. I further declare that the work reported in this project has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

**Place**: Hyderabad

**Date**: 29-09-2020 **Signature of the Candidate**

Contents

[1. INTRODUCTION 5](#_Toc3488918)

[1.1 Steganography 5](#_Toc3488919)

[1.2 Steganography vs. Cryptography: 5](#_Toc3488920)

[1.3 Types of steganography 5](#_Toc3488921)

[1.3.1 Spatial domain Techniques: 5](#_Toc3488922)

[1.3.2 Transform domain techniques: 6](#_Toc3488923)

[1.4 Challenges in Steganography 7](#_Toc3488924)

[1.5 Objective of the project 7](#_Toc3488925)

[1.6 Tool Used: 7](#_Toc3488926)

[1.7 Achievements 9](#_Toc3488932)

[2. DETAILED MODULE DESCRIPTIONS 10](#_Toc3488933)

[2.1 JPEG Compression 10](#_Toc3488934)

[2.2 JPEG Steganography 15](#_Toc3488935)

[2.2.1 Jpeg-Jsteg: 16](#_Toc3488936)

[2.2.2 Steg-Hide: 18](#_Toc3488937)

[3. SAMPLE CODE 21](#_Toc3488938)

[3.1 Jpeg-Jsteg Method 21](#_Toc3488939)

[3.1.1 Encryption 21](#_Toc3488940)

[3.1.2 Decryption 25](#_Toc3488941)

[3.2 Steg-Hide Method 26](#_Toc3488942)

[3.2.1 Encryption 26](#_Toc3488943)

[3.2.2 Decryption 28](#_Toc3488944)

[4. SCREEN SHOTS 30](#_Toc3488945)

[4.1 Encryption 30](#_Toc3488946)

[4.2 Decryption 36](#_Toc3488947)

# 1. INTRODUCTION

## 1.1 Steganography

Steganography is the art of hiding information within innocuous cover carriers in ways such that the hidden message is undetectable. In Greek, „stego‟ means „covered‟ or „secret‟ and „graphy‟ means „to write‟ and therefore, steganography becomes “covered or secret writing”. The information to be hidden is embedded into the cover object which can be a text matter, some image, or some audio/video file in such a way that the very existence of the message is undetected by maintaining the appearance of the resulted object exactly same as the original. The main goal of steganography is to hide the fact that the message is present in the transmission medium.

## 1.2 Steganography vs. Cryptography:

Cryptography is the science of encrypting data in such a way that one cannot understand the encrypted message, whereas in steganography the mere existence of data is concealed, such that even its presence cannot be noticed. Using cryptography might raise some suspicion whereas in steganography the existence of secret message is invisible and thus not known. We can think of steganography as an extension of cryptography.

## 1.3 Types of steganography

On the basis of cover object steganography may be of many types like Audio Steganography, Video Steganography, image Steganography etc. Image Steganography is very popular because of popularity of digital image transmission over the internet. Image Steganography use redundancy of digital image to hide the secret data. It may be divided into two categories. They are spatial-domain methods and frequency-domain ones. In the spatial domain, the secret messages are embedded in the image pixels directly. In the frequency-domain, however, the secret image is first transformed to frequency-domain, and then the messages are embedded in the transformed.

### 1.3.1 Spatial domain Techniques:

**Least Significant Bit (LSB) substitution**- LSB substitution method, which uses fixed k LSBs in each pixel to embed secret message, is the easiest method to hide message in an image. However, it is easy to reveal a stego-image produced by the LSB insertion method.

**Distortion technique**- In distortion technique some pixel property of cover image is changed according to secret message and then deflection of distorted from original image contains secret information.

### 1.3.2 Transform domain techniques:

If we embed information in spatial domain, it may be subjected to the losses if the image undergoes any image processing technique like compression, cropping etc. To overcome this problem we embed the information in frequency domain such that the secret information is embedded on the significant frequency values while higher frequency part is omitted. We first apply transformations to the image then data is to be hidden by changing the values of the transformation coefficients accordingly.

There are mainly three transformation techniques:

**Fast Fourier Transform (FFT) Steganography**- In this technique two dimensional FFT is used to convert the cover image into transform domain first and then secret bits are embedded on the significant coefficients. FFT includes complex term also, hence it computes more mathematical computations and time complexity is higher than DCT steganography.

**Discrete Cosine Transform (DCT) Steganography**- In this technique two dimensional DCT is used for transformation of cover image. DCT is derived from the FFT, however it requires fewer multiplications than the FFT since it works only with real numbers. Also, the DCT produces fewer significant coefficients in its result, which leads to greater compression. Hence DCT is the popular technique in the field of steganography.

If after DCT transformation, quantization step is also taken as in Joint Photographic Experts Group (JPEG) compression then it becomes robust to JPEG compression and this technique is called as JPEG steganography.

**Discrete Wavelet Transform (DWT) Steganography**- In this technique DWT is used to separate high frequency components from low frequency components and then replacing the high frequency part by secret data. The embedding capacity of this technique is far greater than DCT steganography.

## 1.4 Challenges in Steganography

The major challenges of effective steganography are:-

**Security of Hidden Communication**: In order to avoid raising the suspicions of eavesdroppers, while evading the meticulous screening of algorithmic detection, the hidden contents must be invisible both perceptually and statistically. Steganography techniques should produce high imperceptible Stego-image.

**Size of Payload:** Unlike watermarking, which needs to embed only a small amount of copyright information, steganography aims at hidden communication and therefore usually requires sufficient embedding capacity. Requirements for higher payload and secure communication are often contradictory. Depending on the specific application scenarios, a tradeoff has to be sought.

**Robustness**: Stego-image should provide robustness to image processing techniques like compression, cropping, resizing etc. i.e. when any of these techniques are performed on stego-image, secret information should not be destroyed completely.

## 1.5 Objective of the project

In this project my task is to implement 2 techniques of spatial domain steganography(LSB substitution) and 2 techniques of JPEG steganography (Jsteg and Steghide) taking secret information as a data file and as an image and to do comparisons between them in terms of embedding capacity and quality of produced stego-image and robustness to attacks.

## 1.6 Tool Used:

MATLAB is used as simulator to implement the techniques of steganography. MATLAB provides highly computing environment and advanced in-built function for image processing.

## 1.7 Achievements

The aim of this project work was to implement various spatial and transform domain steganographic techniques. The following achievements or we can say objectives were achieved as follows:

|  |  |
| --- | --- |
|  | Spatial domain techniques are easy ways to embed information, but they are highly vulnerable to even small cover modifications. |

|  |  |
| --- | --- |
|  | The embedding capacity of Jpeg steganography is very less than spatial domain  techniques. |

|  |  |
| --- | --- |
|  | The spatial domain techniques provide high PSNR, high perceptual quality and high embedding capacity but these not provide robustness. |

|  |  |
| --- | --- |
|  | Transform domain provide robustness while providing very less embedding capacity, low PSNR and low perceptual quality**.** |

# 2. DETAILED MODULE DESCRIPTIONS

In the project JPEG steganography is implemented which use Discrete Cosine Transform to convert image into frequency domain. Before discussing these techniques in detail we first discuss the JPEG compression because JPEG steganography is just the modified version of JPEG compression.

## 2.1 JPEG Compression

The Joint Photographic Experts Group developed the JPEG algorithm in the late 1980‟s and early 1990‟s. to address the problems of that era, specifically the fact that consumer-level computers had enough processing power to manipulate and display full color photographs. However, full color photographs required a tremendous amount of bandwidth when transferred over a network connection and required just as much space to store a local copy of the image. Other compression techniques had major tradeoffs. They had either very low amounts of compression, or major data loss in the image. Thus, the JPEG algorithm was created to compress photographs with minimal data loss and high compression ratios.

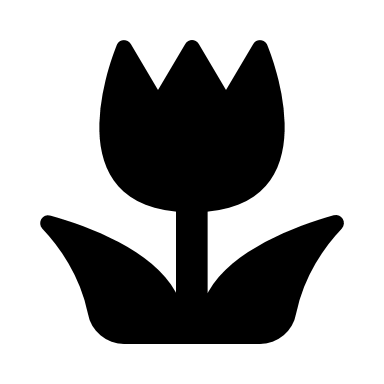
JPEG compression and decompression consist of 4 distinct and independent phases. First, the image is divided into 8 x 8- pixel blocks. Next, a discrete cosine transform is applied to each block to convert the information from the spatial domain to the frequency domain. After that, the frequency information is quantized to remove unnecessary information. Finally, standard compression techniques compress the final bit stream. The basic flow of JPEG is as below:

***JPEG Compression Interface***

DCT

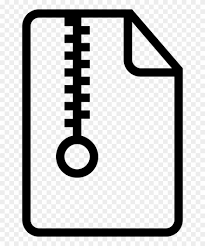
Quantizer

Huffman Coder



Quantization Table

Table



**Source Image**

**8 X 8 Blocks**

**Compressed Image**

Phase One: Divide the Image

Attempting to compress an entire image would not yield optimal results. Therefore, JPEG divides the image into matrices of 8 x 8 pixel blocks. If the image dimensions are not multiples of 8, extra pixels are added to the bottom and right part of the image to pad it to the next multiple of 8 so that we create only full blocks. The dummy values are easily removed during decompression. From this point on, each block of 64 pixels is processed separately from the others, except during a small part of the final compression step.

It may optionally include a change in colorspace. Normally, JPEG will convert RGB colorspace to YCbCr colorspace. In YCbCr, Y is the luminance, which represents the intensity of the color. Cb and Cr are chrominance values, and they actually describe the color itself. YCbCr tends to compress more tightly than RGB.

Phase Two: Conversion to the Frequency Domain

In JPEG all 8 x 8 blocks are converted to frequency domain using DCT. The Discrete Cosine Transform (DCT) is derived from the FFT, however it requires fewer multiplications than the FFT since it works only with real numbers. Also, the DCT produces fewer significant coefficients in its result, which leads to greater compression. That is why DCT is used for transformation purpose. The 2D discrete cosine transform equation is given below

where f (x, y) is the 8-bit image value at coordinates (x, y) and F (u, v) is the new entry in the frequency matrix. Also, C(x) = 1/√ 2 if x is 0, and C(x) = 1 for all other cases.

The frequency domain matrix contains values from -1024…1023. The upper-left entry, also known as the DC value, is the average of the entire block, and is the lowest frequency cosine coefficient. As we move right the coefficients represent cosine functions in the vertical direction that increase in frequency. Likewise, as we move down, the coefficients belong to increasing frequency cosine functions in the horizontal direction. The highest frequency values occur at the lower-right part of the matrix. The higher frequency values also have a natural tendency to be significantly smaller than the low frequency coefficients since they contribute much less to the image. Typically, the entire lower-right half of the matrix is factored out after quantization. This essentially removes half of the data per block, which is one reason why JPEG is so efficient at compression. Computing the DCT is the most time-consuming part of JPEG compression. Thus, it determines the worst-case running time of the algorithm.

Phase Three: Quantization

Having the data in the frequency domain allows the algorithm to discard the least significant parts of the image. The JPEG algorithm does this by dividing each cosine coefficient in the data matrix by some predetermined constant, and then rounding up or down to the closest integer value. These predefined constants are then entered into another 8 x 8 matrix, called the „quantization matrix‟. Each entry in the quantization matrix corresponds to exactly one entry in the frequency matrix. A typical quantization matrix will be symmetrical about the diagonal and will have lower values in the upper left and higher values in the lower right. Since any arbitrary values could be used during quantization, the entire quantization matrix is stored in the final JPEG file so that the decompression routine will know the values that were used to divide each coefficient. The constant values that are used in the division may be arbitrary, although research has determined some very good typical values. The standard quantization matrix used in JPEG is as below:

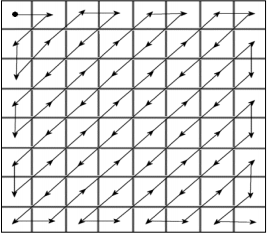
Dividing by a high constant value can introduce more error in the rounding process, but high constant values have another effect. As the constant gets larger the result of the division approaches zero. This is especially true for the high frequency coefficients, since they tend to be the smallest values in the matrix. Thus, many of the frequency values become zero. Phase four takes advantage of this fact to further compress the data.

The algorithm used to calculate the quantized frequency matrix is fairly simple. It takes a value from the frequency matrix (F) and divides it by its corresponding value in the quantization matrix (Q). This gives the final value for the location in the quantized frequency matrix (Fquantize). The quantization equation that is used for each block in the image is below:

By adding 0.5 to each value, we essentially round it off automatically when we truncate it, without performing any comparisons.

Phase Four: Entropy Coding:

After quantization, the algorithm is left with blocks of 64 values, many of which are zero. Of course, the best way to compress this type of data would be to collect all the zero values together, which is exactly what JPEG does. The algorithm uses a zigzag ordered encoding, which collects the high frequency quantized values into long strings of zeros. To perform a zigzag encoding on a block, the algorithm starts at the DC value and begins winding its way down the matrix, as shown in figure. This converts an 8 x 8 table into a 1 x 64 vector



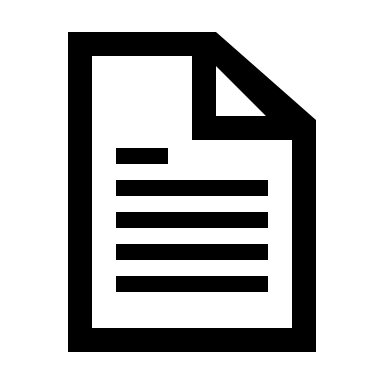
*Zig-Zag Ordered Encoding*

All of the values in each block are encoded in this zigzag order except for the DC value. For all of the other values, there are two tokens that are used to represent the values in the final file. The first token is a combination of {size, skip} values. The size value is the number of bits needed to represent the second token, while the skip value is the number of zeros that precede this token. The second token is simply the quantized frequency value, with no special encoding. At the end of each block, the algorithm places an end- of-block sentinel so that the decoder can tell where one block ends and the next begins.

The first token, with {size, skip} information, is encoded using Huffman coding. Huffman coding scans the data being written and assigns fewer bits to frequently occurring data, and more bits to infrequently occurring data. Thus, if a certain value of size and skip happen often, they may be represented with only a couple of bits each. There will then be a lookup table that converts the two bits to their entire value. JPEG allows the algorithm to use a standard Huffman table, and also allows for custom tables by providing a field in the file that will hold the Huffman table. DC values use delta encoding, which means that each DC value is compared to the previous value, in zigzag order. This is the only instance where blocks are not treated independently from each other. The difference between the current DC value and the previous value is all that is included in the file.

## 2.2 JPEG Steganography

JPEG steganography is more important and popular because stago-image produce by these techniques are robust to jpeg compression. In this technique secret data is embed after quantization phase of JPEG compression. Only significant quantized DCT coefficients are modified according to secret bits. Remaining steps are similar to JPEG compression. In this way stego-image is produced in .jpg format directly. The basic flow diagram of embedding and extraction process is as below:



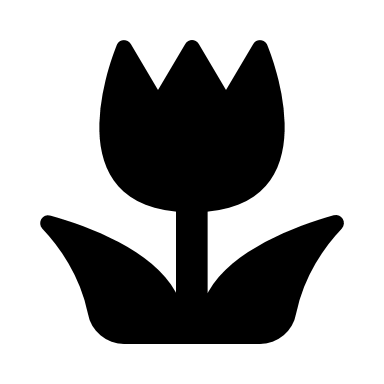
Quantization Table

8 X 8 Block DCT

Quantizer

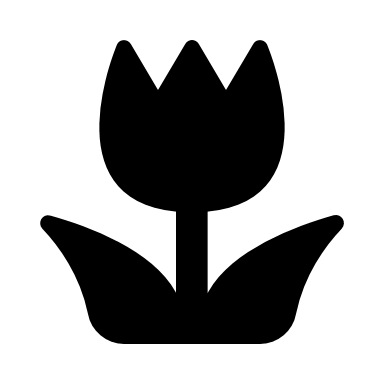
Embedding

Entropy Coding



Secret message into bit stream conversion

Secret Message



Stego Image

Cover Image

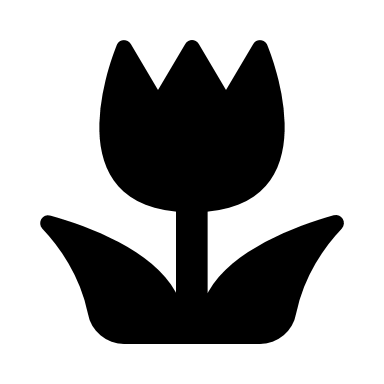
*Block diagram of Embedding process of JPEG Steganography*

8 X 8 Block DCT

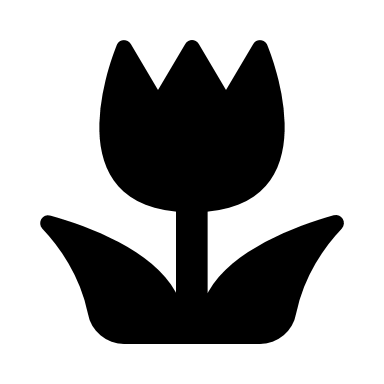
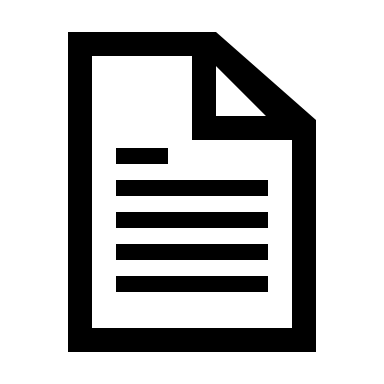
Quantizer

Embedding

Entropy Coding



Secret message in bit stream



Quantization Table

Secret Message

Distorted Cover Image

Stego Image

*Block diagram of Extraction process of JPEG Steganography*

Following are some JPEG steganography techniques:

1. Jsteg 2) Outguess 3) Steghide 4) F3,F4,F5 5) J3

In the project only 2 techniques Jsteg and Steghide is implemented and these are explained below with algorithms.

### 2.2.1 Jpeg-Jsteg:

In Jpeg–Jsteg, the secret messages are embedded in LSB of quantized DCT coefficients whose values are not (0, 1, or 1). Its execution steps are described briefly as follows. First, JPEG partitions a cover-image into non-overlapping blocks of 8\*8 pixels, and then it uses DCT to transform each block into DCT coefficients. The results of the DCT coefficients are scaled according to a quantization table.

The standard quantization table is listed in Fig., which is a matrix that contains 64 coefficients. Next, Jpeg–Jsteg uses an encryption algorithm to protect the message. A message after encrypting is called secret message S={s1, s2, s3, s4, . . .,sn }, where si is a secret bit. After the above steps, Jpeg–Jsteg embeds si into LSB of quantize DCT coefficients whose values are not 0, 1, or 1. The bit-flip diagram of J-steg is as below:

JPEG Coefficients

in cover image -4 -3 -2 -1 0 0 1 2 3 4

0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1

JPEG coefficients

in Stego-image

*Bit-flip in JPEG-JSTEG*

The embedding sequence employed in Jpeg–Jsteg is in the zigzag scan order, which is listed in fig. After embedding the secret message in each block, Jpeg–Jsteg uses Huffman coding, Run-Length coding, and DPCM of JPEG entropy coding to compress each block. Finally, Jpeg–Jsteg obtains a JPEG stego-image. The message capacity of Jpeg–Jsteg is limited. If there are many quantifized coefficients equal to 0, 1, or 1, then the message capacity of Jpeg–Jsteg will be decreased. Besides, in DCT transformation, most important coefficients are located around the low-frequency part. Jpeg– Jsteg modifies the quantized DCT coefficients right in the low-frequency part. Therefore, the image quality of Jpeg–Jsteg is degraded, especially when the cover-image undergoes a high compression ratio.

Embedding Algorithm:

Input: secret message, cover image

Procedure:

**Step1:** Convert the secret message into bit stream

**Step2:** Divide the cover image into 8x8 blocks

**Step3:** Calculate DCT coefficients for each block

**Step4:** Quantize the coefficients

**Step5:** while complete message not embedded do

{get next DCT coefficient

if DCT ≠ 0, DCT ≠1 and DCT ≠ -1 then

{get next bit from message

replace DCT LSB with message bit}

}

**Step6:** De-quantize and take inverse DCT to obtain stego-image. End.

Output: Stego- image

Extracting Algorithm:

Input: Stego image

Procedure:

**Step1:** Divide the stego image into 8x8 blocks

**Step2:** Calculate DCT coefficients for each block

**Step3:** Quantize the coefficients

**Step4:** While secret message not completed do

{ Get next DCT coefficient

If DCT ≠ 0 , DCT ≠1and DCT ≠ -1 then

{ Concatenate DCT LSB to secret message bit steam } }

**Step5:** Convert secret message bit stream into the message. End.

Output: Secret message

***Advantage:***

* Robust to JPEG compression.
* Less bandwidth requirement for stego image transmission because its size can be reduced.

***Disadvantage:***

* Only few secret messages can be embedded in cover image.
* High detectability due to the fact that it introduces characteristic artifacts into the first order statistics (histogram) of DCT coefficients.

### 2.2.2 Steg-Hide:

Steghide uses a graph-theoretic approach to steganography. Steghide embeds by swapping DCT coefficients and thus avoids changing the histogram. The fact that the embedding is done by exchanging pixel values implies that the first-order statistics (i.e. the number of times a colour occurs in the picture) is not changed. During the encoding process, the sender splits the cover image in 8 x 8-pixel blocks; each block encodes exactly one secret message bit. The embedding process starts with selecting a pseudorandom block bi which will be used to code the ith message bit. Before the communication starts, both sender and receiver have to agree on the location of two DCT coefficients, which will be used in the embedding process; let us denote these two indices by (u1,v1) and (u2, v2). The two coefficients should correspond to cosine functions with middle frequencies; this ensures that the information is stored in significant parts of the signal (hence the embedded information will not be completely damaged by JPEG compression).

Embedding Algorithm:

Input: secret message, cover image

Procedure:

**Step1:** Convert the secret message into bit stream

**Step2:** Divide the cover image into 8x8blocks

**Step3:** Calculate DCT coefficients for each block

**Step4:** Quantize the coefficients

**Step5:** While complete message not embedded do

{ Get next block

If block(u1,v1) ≠ block(u2,v2)

then { get next bit from message stream

if bit =1 and block(u1,v1)< block(u2,v2)

{ swap(block(u1,v1),block(u2,v2) }

If bit =0 and block(u1,v1)> block(u2,v2)

{ swap(block(u1,v1),block(u2,v2) } } }

**Step6:** De-quantize and take inverse DCT to obtain stego-image End.

Output: Stego- image

Extracting Algorithm:

Input: Stego image

Procedure:

**Step1:** Divide the stego image into 8x8 blocks

**Step2:** Calculate DCT coefficients for each block

**Step3:** Quantize the coefficients

**Step4:** While secret message not completed do

{ Get next block

If block(u1,v1) ≠ block(u2,v2)

then { If block(u1,v1)< block(u2,v2)

{ concatenate 0 to bit stream Else Concatenate 0 to bit stream } }

}

**Step5:** Convert secret message bit stream into the message. End.

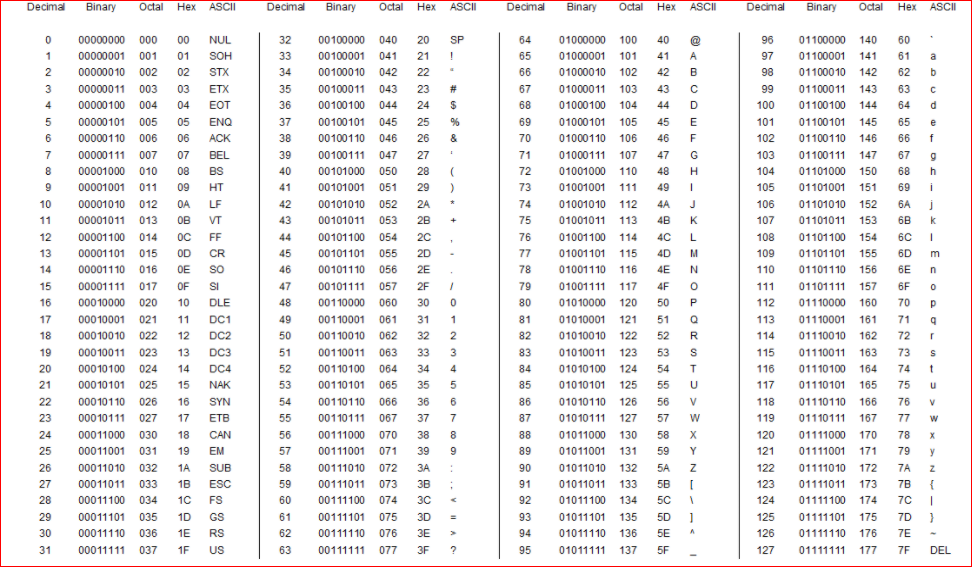
Output: Secret message

***Advantage:***

* It cannot be detected by steganalysis which uses first order characteristics, because histogram is preserved.
* It provides Robustness to JPEG compression.

***Disadvantage:***

* Its embedding capacity is less than Jsteg too.
* Cover size in which message embedded is always multiple of 8 which can be used to detect the presence of secret message.



# 3. SAMPLE CODE

#import all the required libraries

import cv2 #OpenCV-Python is a library of Python bindings designed to solve computer vision problems.

import numpy as np #numerical Python library. NumPy contains a multi-dimensional array and matrix data structures.

import types

from matplotlib import pyplot as plt #used for displaying the image

def messageToBinary(message):

#normal string to binary format

if type(message) == str:

return ''.join([ format(ord(i), "08b") for i in message ])

elif type(message) == bytes or type(message) == np.ndarray:

return [ format(i, "08b") for i in message ]

elif type(message) == int or type(message) == np.uint8:

return format(message, "08b")

else:

raise TypeError("Input type not supported")

# Function to hide the secret message into the image

def hideData(image, secret\_message):

#Threshold value check

# calculate the maximum bytes to encode

n\_bytes = image.shape[0] \* image.shape[1] \* 3 // 8

print(image.shape[0])

print(image.shape[0])

print("Maximum bytes to encode:", n\_bytes)

#Check if the number of bytes to encode is less than the maximum bytes in the image

if len(secret\_message) > n\_bytes:

raise ValueError("Error encountered insufficient bytes, need bigger image or less data !!")

secret\_message += "#####" # you can use any string as the delimeter

data\_index = 0

# convert input data to binary format using messageToBinary() fucntion

binary\_secret\_msg = messageToBinary(secret\_message)

data\_len = len(binary\_secret\_msg) #Find the length of data that needs to be hidden

for values in image:

for pixel in values:

# convert RGB values to binary format

#print(pixel)

r, g, b = messageToBinary(pixel)

#print(values)

print(pixel)

# modify the least significant bit only if there is still data to store

if data\_index < data\_len:

# hide the data into least significant bit of red pixel

pixel[0] = int(r[:-1] + binary\_secret\_msg[data\_index], 2)

data\_index += 1

if data\_index < data\_len:

# hide the data into least significant bit of green pixel

pixel[1] = int(g[:-1] + binary\_secret\_msg[data\_index], 2)

data\_index += 1

if data\_index < data\_len:

# hide the data into least significant bit of blue pixel

pixel[2] = int(b[:-1] + binary\_secret\_msg[data\_index], 2)

data\_index += 1

# if data is encoded, just break out of the loop

if data\_index >= data\_len:

break

return image

# Function to hide the secret message into the image

def hideData(image, secret\_message):

#Threshold value check

# calculate the maximum bytes to encode

n\_bytes = image.shape[0] \* image.shape[1] \* 3 // 8

print(image.shape[0])

print(image.shape[0])

print("Maximum bytes to encode:", n\_bytes)

#Check if the number of bytes to encode is less than the maximum bytes in the image

if len(secret\_message) > n\_bytes:

raise ValueError("Error encountered insufficient bytes, need bigger image or less data !!")

secret\_message += "#####" # you can use any string as the delimeter

data\_index = 0

# convert input data to binary format using messageToBinary() fucntion

binary\_secret\_msg = messageToBinary(secret\_message)

data\_len = len(binary\_secret\_msg) #Find the length of data that needs to be hidden

for values in image:

for pixel in values:

# convert RGB values to binary format

#print(pixel)

r, g, b = messageToBinary(pixel)

#print(values)

print(pixel)

# modify the least significant bit only if there is still data to store

if data\_index < data\_len:

# hide the data into least significant bit of red pixel

pixel[0] = int(r[:-1] + binary\_secret\_msg[data\_index], 2)

data\_index += 1

if data\_index < data\_len:

# hide the data into least significant bit of green pixel

pixel[1] = int(g[:-1] + binary\_secret\_msg[data\_index], 2)

data\_index += 1

if data\_index < data\_len:

# hide the data into least significant bit of blue pixel

pixel[2] = int(b[:-1] + binary\_secret\_msg[data\_index], 2)

data\_index += 1

# if data is encoded, just break out of the loop

if data\_index >= data\_len:

break

return image

def showData(image):

binary\_data = ""

for values in image:

for pixel in values:

r, g, b = messageToBinary(pixel) #convert the red,green and blue values into binary format

binary\_data += r[-1] #extracting data from the least significant bit of red pixel

binary\_data += g[-1] #extracting data from the least significant bit of red pixel

binary\_data += b[-1] #extracting data from the least significant bit of red pixel

# split by 8-bits

all\_bytes = [ binary\_data[i: i+8] for i in range(0, len(binary\_data), 8) ]

# convert from bits to characters

decoded\_data = ""

for byte in all\_bytes:

decoded\_data += chr(int(byte, 2))

if decoded\_data[-5:] == "#####": #check if we have reached the delimeter which is "#####"

break

#print(decoded\_data)

return decoded\_data[:-5] #remove the delimeter to show the original hidden message

# Encode data into image

def encode\_text():

image\_name = input("Enter image name(with extension): ")

image = cv2.imread(image\_name) # Read the input image using OpenCV-Python.

#It is a library of Python bindings designed to solve computer vision problems.

#details of the image

print("The shape of the image is: ",image.shape) #check the shape of image to calculate the number of bytes in it

print("The original image is as shown below: ")

plt.imshow(image)

resized\_image = cv2.resize(image, (500, 500)) #resize the image as per your requirement

data = input("Enter data to be encoded : ")

if (len(data) == 0):

raise ValueError('Data is empty')

filename = input("Enter the name of new encoded image(with extension): ")

encoded\_image = hideData(image, data) # call the hideData function to hide the secret message into the selected image

cv2.imwrite(filename, encoded\_image)

print("Scuccessfully encrypted the data to the image!!!!!!!!!!!! ")

# Decode the data in the image

def decode\_text():

# read the image that contains the hidden image

image\_name = input("Enter the name of the steganographed image that you want to decode (with extension) :")

image = cv2.imread(image\_name) #read the image using cv2.imread()

print("The Steganographed image is as shown below: ")

resized\_image = cv2.resize(image, (500, 500)) #resize the original image as per your requirement

#cv2.imshow(resized\_image) #display the Steganographed image

text = showData(image)

return text

# Image Steganography

def Steganography():

a = input("Image Steganography \n 1. Encode the data \n 2. Decode the data \n Your input is: ")

userinput = int(a)

if (userinput == 1):

print("\nEncoding....")

encode\_text()

elif (userinput == 2):

print("\nDecoding....")

print("Decoded message is " + decode\_text())

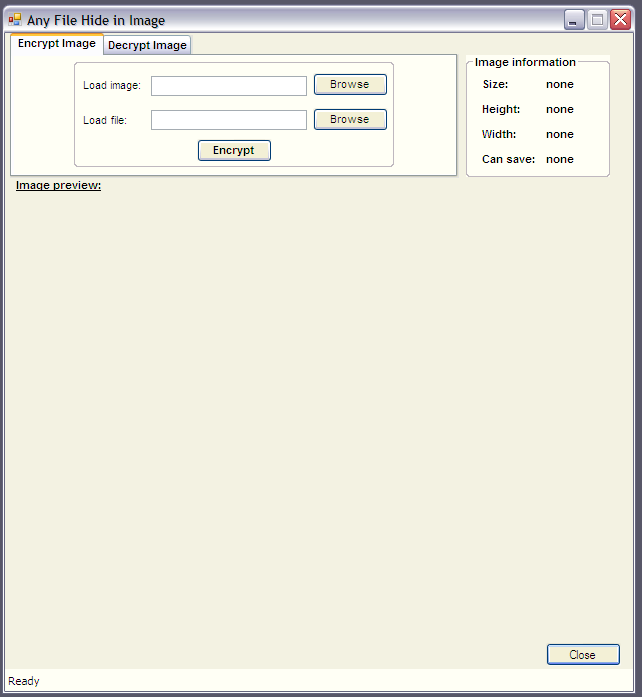
else:

raise Exception("Enter correct input")

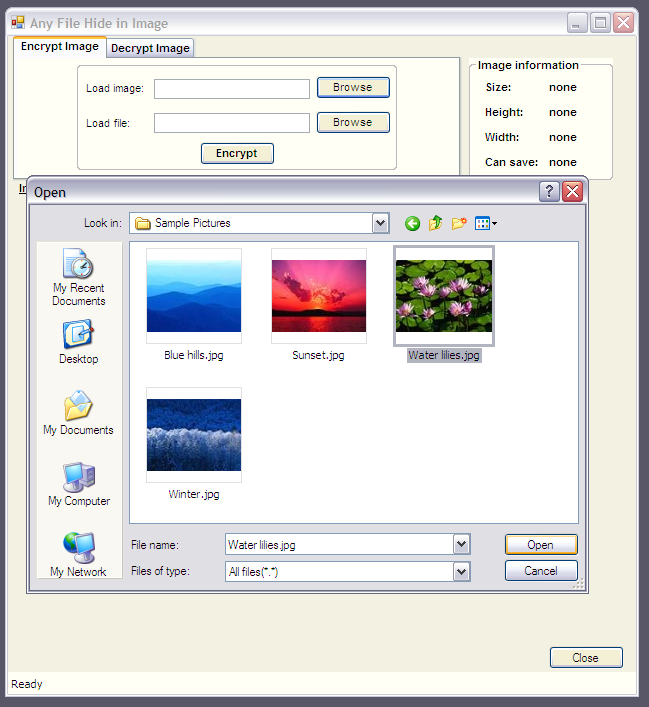
Steganography() #encode image

# 4. SCREEN SHOTS

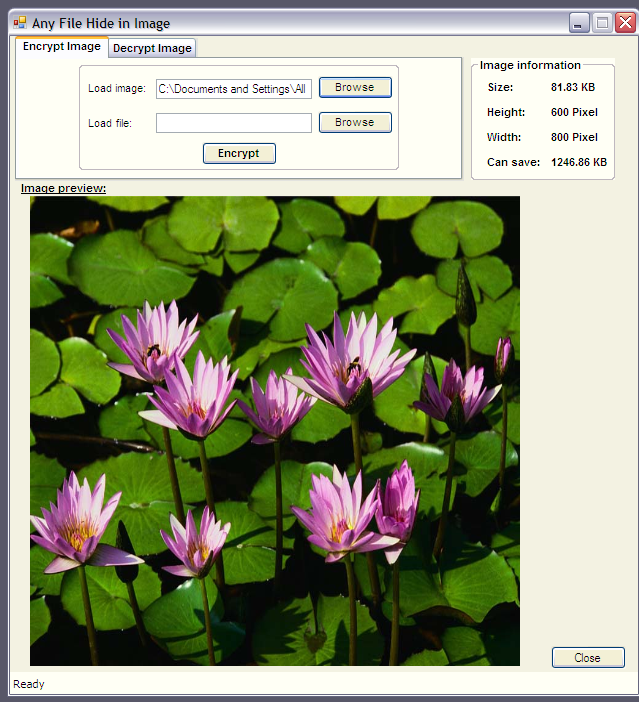
## 4.1 Encryption



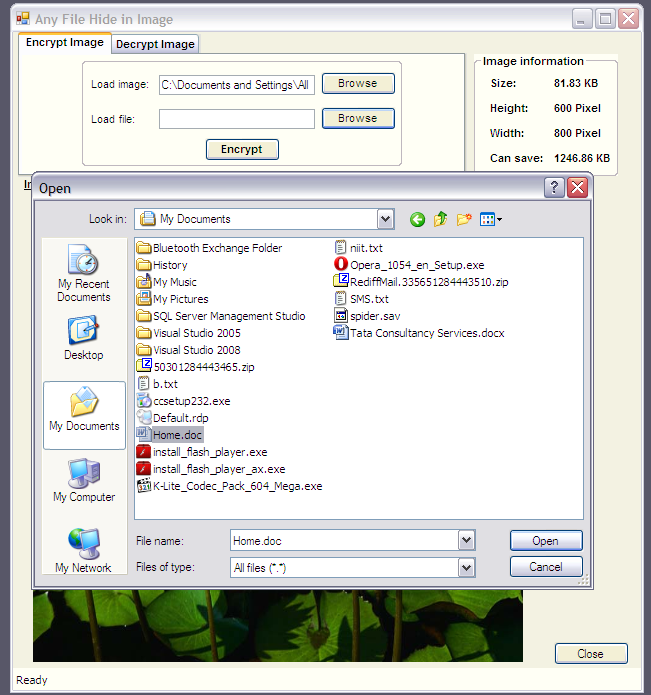
*1.For Encryption select Encrypt Image tab option*



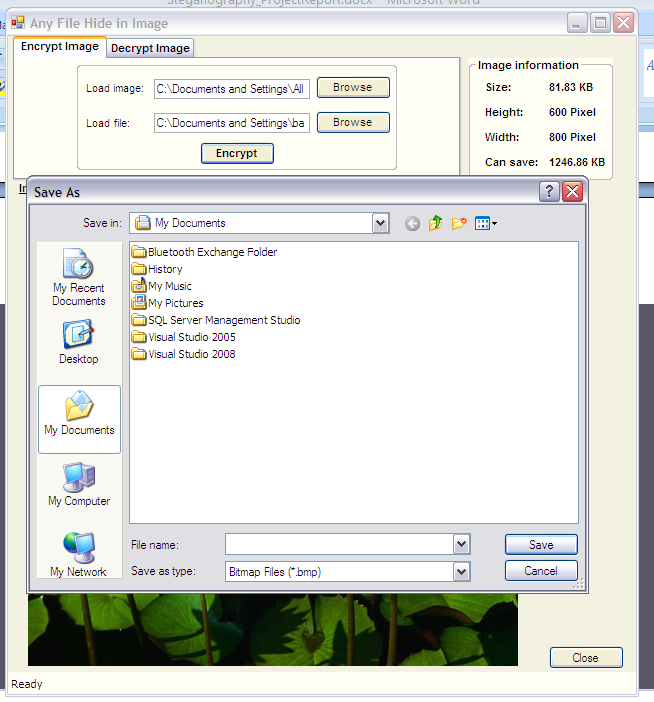
*2.For load image click on button “Browse” that is next to the Load Image textbox. The file open dialog box will displays as follows, select the Image file, which you want to use hide information and click on Open button.*



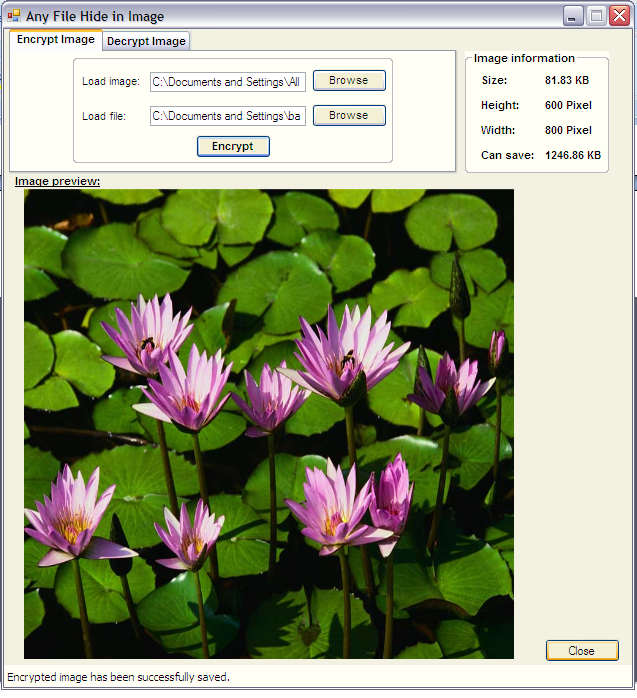
*3. The image file will opened and is displays as follows. Next, click on “Browse” button that is next to the Load File textbox.*



*4. Again the file open dialog box will appear, select any type of file whatever you want to hide with the image and click on ok button.*

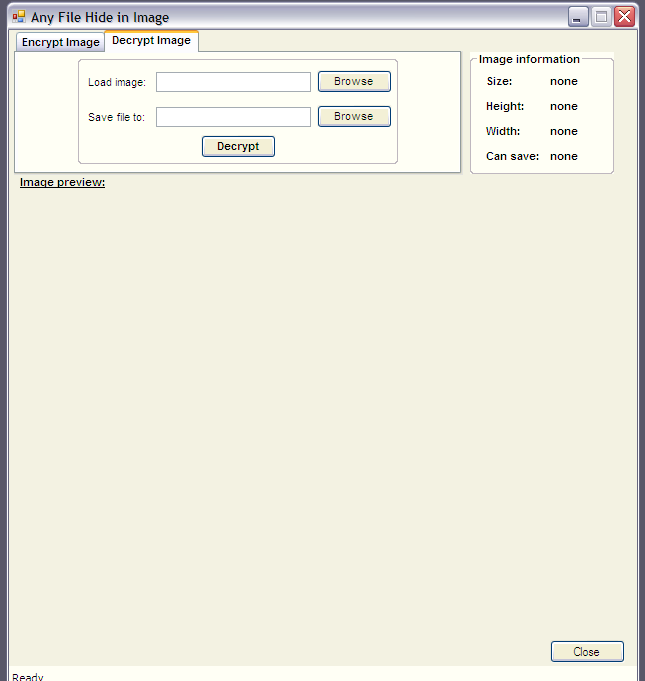


*5. The next step is to encrypt the file. Now click on “Encrypt” button, it will open the save dialog box which ask you to select the path to save the New image file and the Image file name. The default format of image file is BMP.*

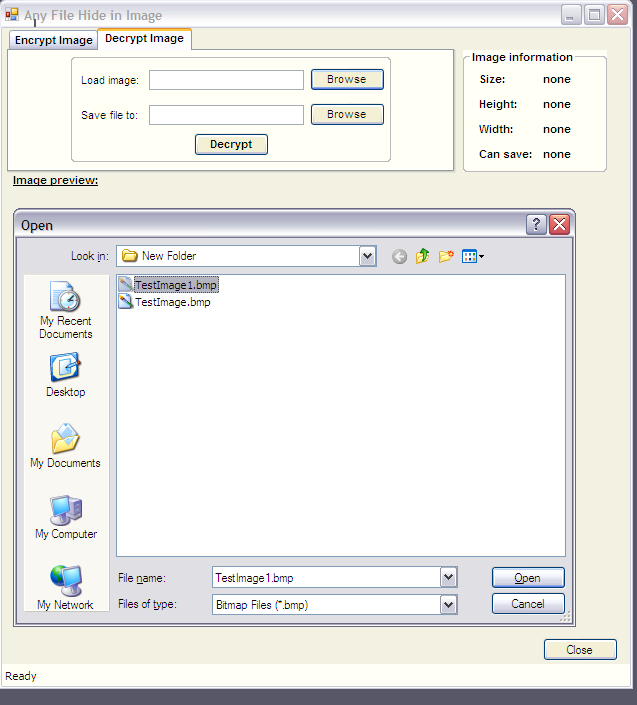


*6.Click encryption to encrypt image*

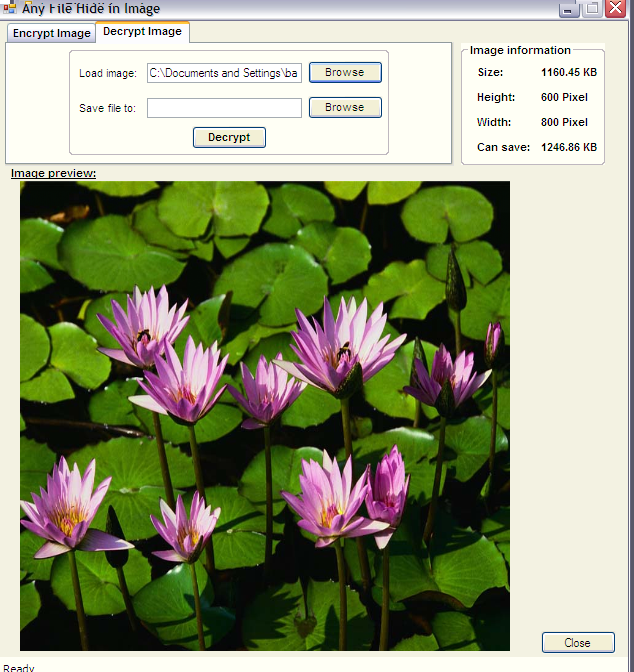
## 4.2 Decryption



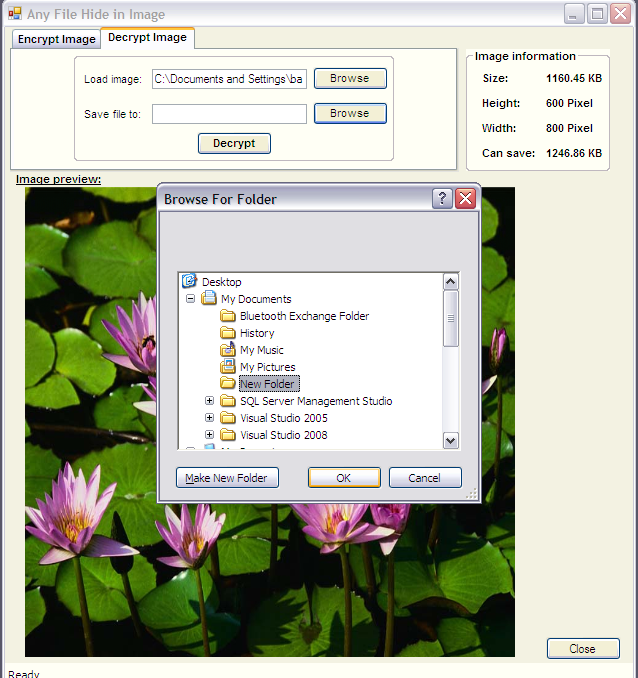
1. *Select the Decryption Image tab option*



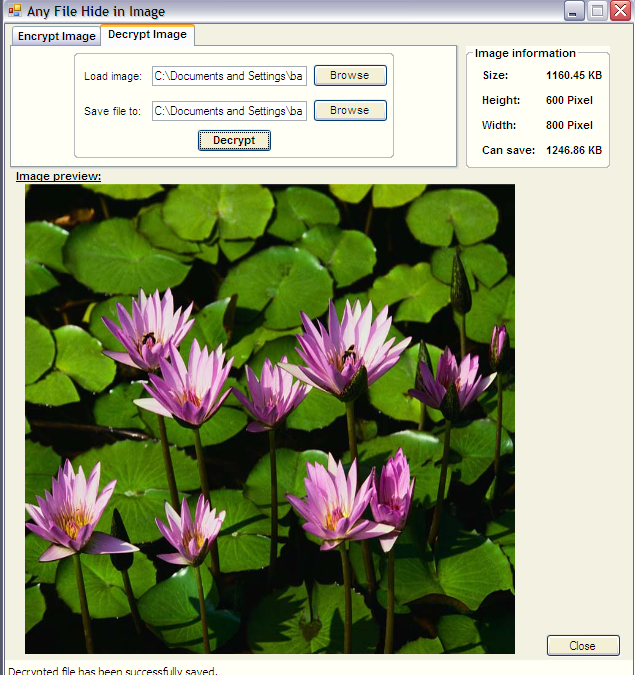
1. *Next click on the “Browse” button, which open the Open file dialog box, here you have to select the image which is Encrypted and has hidden information file. Select the image file and click on Open button.*



1. *The image file displayed as follows:*



1. *Now click on “Browse” button which is next to “Save file to” textbox. It will open a dialog box that is “Browse for folder”. It ask you to select the path or folder, where you want to extract the hidden file. Select the folder and click on Ok button.*



1. *Now click on Decrypt button, it will decrypt the image, the hidden file and image file is saved into selected folder. The message for successful decryption is displayed on the status bar which is places at bottom of the screen.*