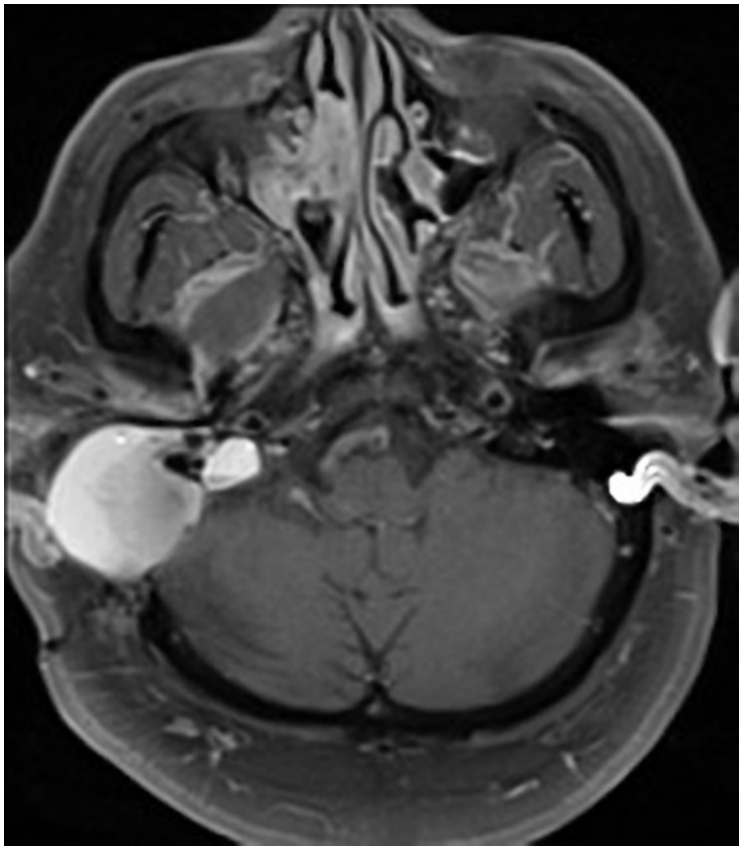
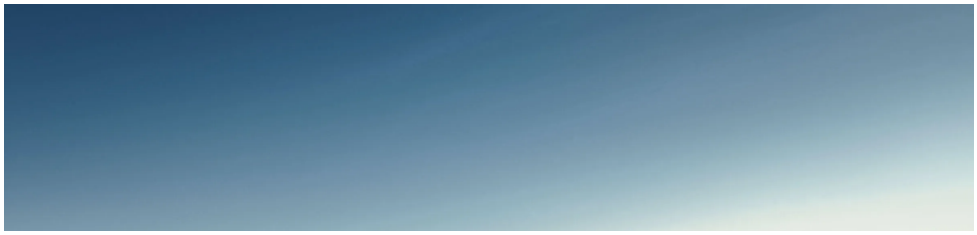
```
from PIL import Image


image1 = Image.open('brain.jpg')
image2=Image.open('super car.jpg')


image1
```



```
image2
```



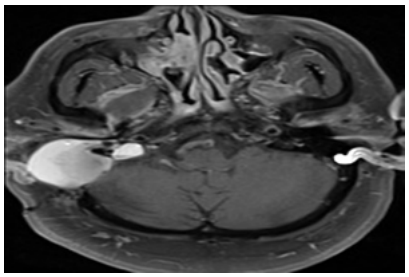
```
image1.size
```

```
(551, 630)
```

Resizing Image 1

```
image1= image1.resize((300,200))
```

```
image1
```



```
image1.size
```

```
(300, 200)
```

```
image2.size
```

```
(2400, 1800)
```

We will be merging image 1 on image 2

Integer to binary Coversion->

```
def int2bin(rgb):

    #Will convert RGB pixel values from integer to binary
    #INPUT: An integer tuple (e.g. (220, 110, 96))
    #OUTPUT: A string tuple (e.g. ("00101010", "11101011", "00010110"))

    r, g, b = rgb
    return ('{0:08b}'.format(r),
            '{0:08b}'.format(g),
            '{0:08b}'.format(b))
    #Return converted r,g,b binary values separately..
```

Binary to Integer Conversion-->

```
def bin2int(rgb):

    #Will convert RGB pixel values from binary to integer.
    #Reverse of the first part.

    r, g, b = rgb
    return (int(r, 2),
            int(g, 2),
            int(b, 2))
    #return converted r,g,b integer values separately


r,g,b=int2bin((225,6,7))
r
```

```
'11100001'
```

```
r[:4]
```

```
'1110'
```

[:4] will be taking the first 4 digits.

And [:-4] will be taking the digits ignoring the last 4 digits..

```
print(g)
print(b)
```

```
00000110
00000111
```

```
bin2int(('11100001', '00000110', '00000111'))
```

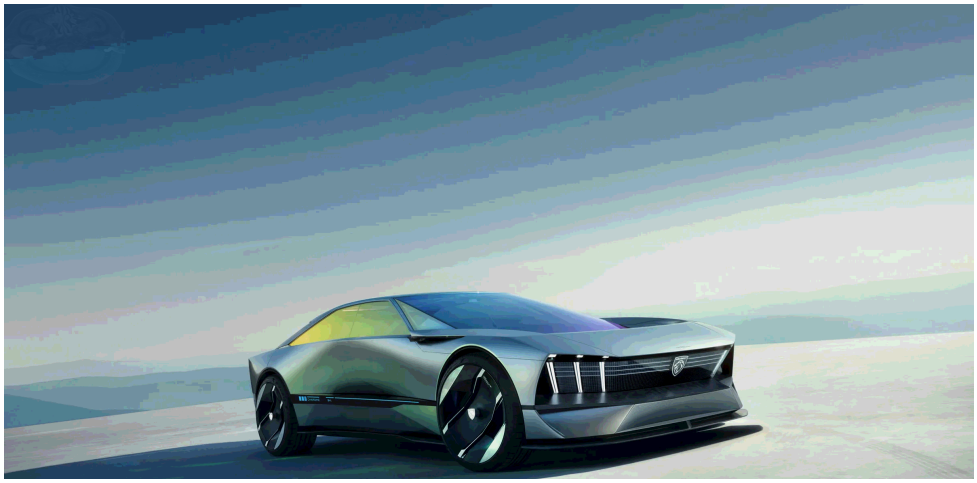
```
(225, 6, 7)
```

Our convertion functions are working perfectly.

```
def merge2rgb(rgb1,rgb2):

    #Will merge two RGB pixels using 4 least significant bits.
    #INPUT: A string tuple ( ("00101010", "11101011", "00010110")),another string tuple (e.g. ("00101010", "11101011", "00010110"))
    #OUTPUT: An integer tuple with the two RGB values merged
    #Will be merging the first four digits of first image and first four digits of 2nd image(i.e to be merged) as last four digits..


    r1,g1,b1=rgb1
    r2,g2,b2=rgb2

    return (r1[:4]+r2[:4],
             g1[:4]+g2[:4],
             b1[:4]+b2[:4]
           )
```

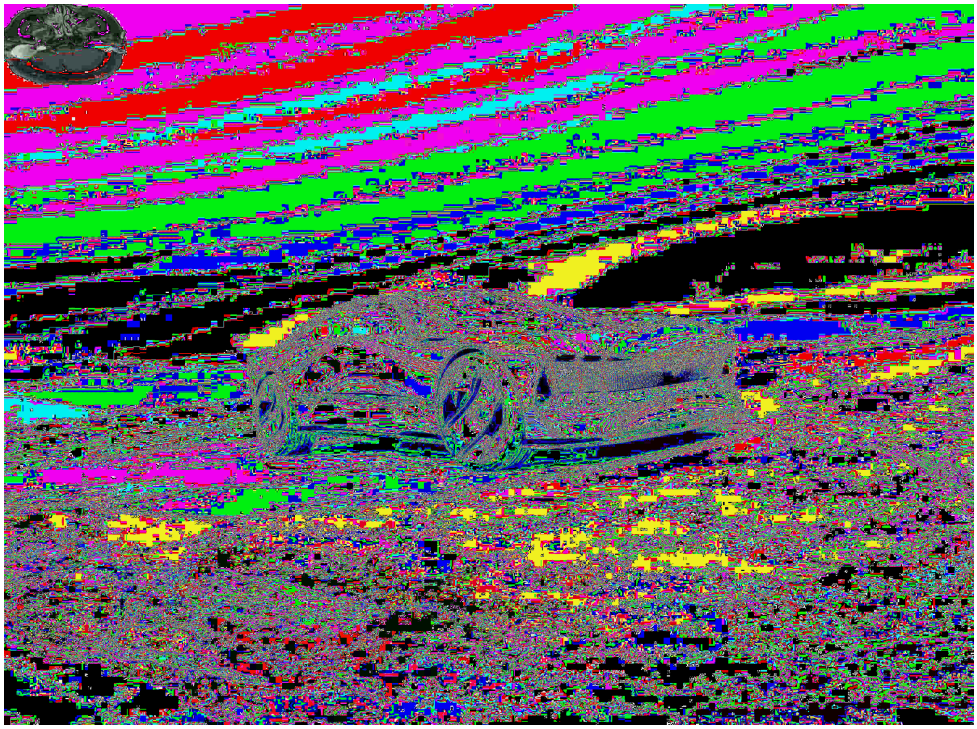
Function to merge two Images-->

```
def merge2img(img1,img2):
  # The First image will be merged into the second image.


  image1=img1
  image2=img2
  #print('rahul')

  # Condition for merging
  if(image1.size[0]>image2.size[0] or image1.size[1]>image2.size[1]):
    print("Cannot merge as the size of 1st Image is greater than size of 2nd Image")
    return
  # Getting the pixel map of the two images

  pixel_tuple1 = image1.load()
  pixel_tuple2 = image2.load()

  #print(pixel_tuple1)
  #print(pixel_tuple2)

  # The new image that will be created.
  new_image = Image.new(image2.mode, image2.size) # Setting the size of Image 2 as Image 1 will be merged to Image 2.
  pixels_new = new_image.load()

  for row in range(image2.size[0]):
    for col in range(image2.size[1]):

        rgb1 = int2bin(pixel_tuple2[row, col])

        # Using a black pixel as default
        rgb2 = int2bin((0, 0, 0))

        # Converting the pixels of image 1 if condition is satisfied

        if(row <image1.size[0] and col< image1.size[1]):
          rgb2= int2bin(pixel_tuple1[row,col])


        merge_rgb= merge2rgb(rgb1,rgb2)

        pixels_new[row,col] = bin2int(merge_rgb)

  #print('rahul')
  new_image.convert('RGB').save('merged1.jpg')

  return new_image




image1 = image1.convert("RGB")
image2=image2.convert("RGB")


 merge2img(image1,image2)
```



See now our image 1 is merged inside image 2.But still image 2 is looking as it was earlier..

```
def unmerge(path):

    img=Image.open(path)

    # Loading the pixel map
    pixel_map = img.load()

    new_image = Image.new(img.mode, img.size)
    pixels_new = new_image.load()

    # Tuple used to store the image original size
    original_size = img.size

    for row in range(img.size[0]):
        for col in range(img.size[1]):
            # Get the RGB (as a string tuple) from the current pixel
            r, g, b = int2bin(pixel_map[row, col])

            # Extract the last 4 bits (corresponding to the hidden image)
            # Concatenate 4 zero bits because we are working with 8 bit values
            rgb = (r[4:] + "0000",
                   g[4:] + "0000",
                   b[4:] + "0000")

            # Convert it to an integer tuple
            pixels_new[row, col] = bin2int(rgb)

            # If this is a 'valid' position, store it
            # as the last valid position
            if pixels_new[row, col] != (0, 0, 0):
                original_size = (row + 1, col + 1)

    # Crop the image based on the 'valid' pixels
    new_image = new_image.crop((0, 0, original_size[0], original_size[1]))

    new_image.save('unmerged1.png')

    return new_image


unmerge('/content/merged1.jpg')
```



Double-click (or enter) to edit

Here we are seeing that the unmerged image is not clear at all

now we should change our merging pattern.

We can now take 2 MSBs from image 1 and add 6 MSBs of image2 while merging..

```
def merge2rgb2(rgb1, rgb2):

    r1, g1, b1 = rgb1
    r2, g2, b2 = rgb2
    rgb = (r1[:6] + r2[:2],
           g1[:6] + g2[:2],
           b1[:6] + b2[:2])
    return rgb
```

```
def merge2img2(img1, img2):

    image1=img1
    image2=img2
    #print('rahul')

    # Condition for merging
    if(image1.size[0]>image2.size[0] or image1.size[1]>image2.size[1]):
       print("Cannot merge as the size of 1st Image is greater than size of 2nd Image")
       return

  # Getting the pixel map of the two images
    pixel_tuple1 = image1.load()
    pixel_tuple2 = image2.load()

    #print(pixel_tuple1)
    #print(pixel_tuple2)

    # The new image that will be created.
    new_image = Image.new(image2.mode, image2.size) # Setting the size of Image 2 as Image 1 will be merged to Image 2.
    pixels_new = new_image.load()

    for row in range(image2.size[0]):
      for col in range(image2.size[1]):

        rgb1 = int2bin(pixel_tuple2[row, col])

        # Using a black pixel as default
        rgb2 = int2bin((0, 0, 0))

        # Converting the pixels of image 1 if condition is satisfied

        if(row <image1.size[0] and col< image1.size[1]):
          rgb2= int2bin(pixel_tuple1[row,col])


        merge_rgb= merge2rgb2(rgb1,rgb2)

        pixels_new[row,col] = bin2int(merge_rgb)

    #print('rahul')
    new_image.convert('RGB').save('merged2.jpg')

    return new_image


def unmerge2(img):

    pixel_map = img.load()

    new_image = Image.new(img.mode, img.size)
    pixels_new = new_image.load()


    original_size = img.size

    for row in range(img.size[0]):
        for col in range(img.size[1]):
            r, g, b = int2bin(pixel_map[row, col])

            # Extracting the last 6 bits (corresponding to the hidden image) and adding zeroes to increase the brightness.

            rgb = (r[6:] + "000000",
                   g[6:] + "000000",
                   b[6:] + "000000")

            # Convert it to an integer tuple
            pixels_new[row, col] = bin2int(rgb)

           #If this is a 'valid' position, store it as a last valid option
            if pixels_new[row, col] != (0, 0, 0):
                original_size = (row + 1, col + 1)

    # Crop the image based on the 'valid' pixels
    new_image = new_image.crop((0, 0, original_size[0], original_size[1]))

    return new_image


merged_image2 = merge2img2(image1,image2)
merged_image2
```

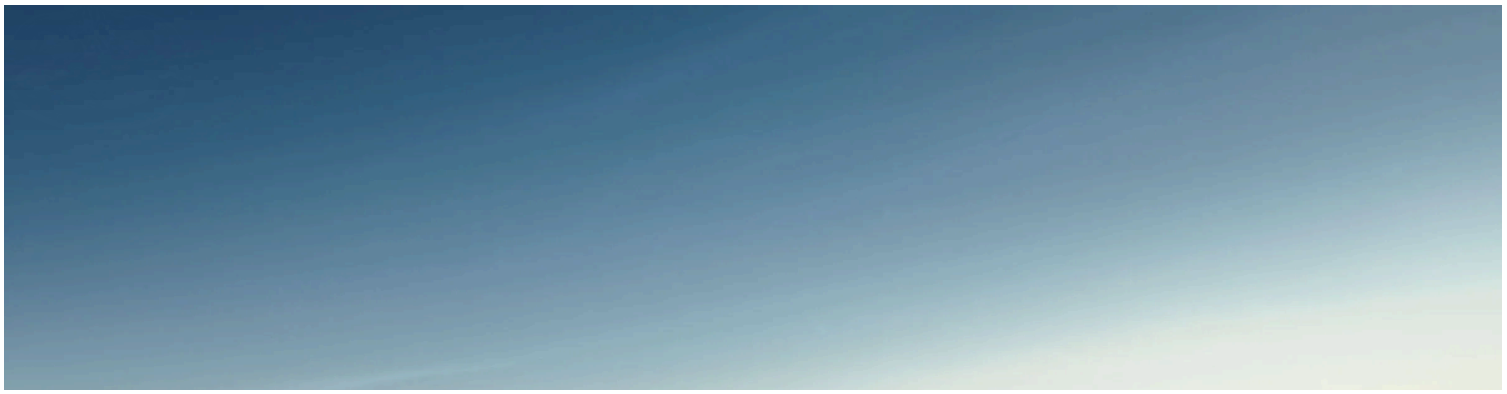
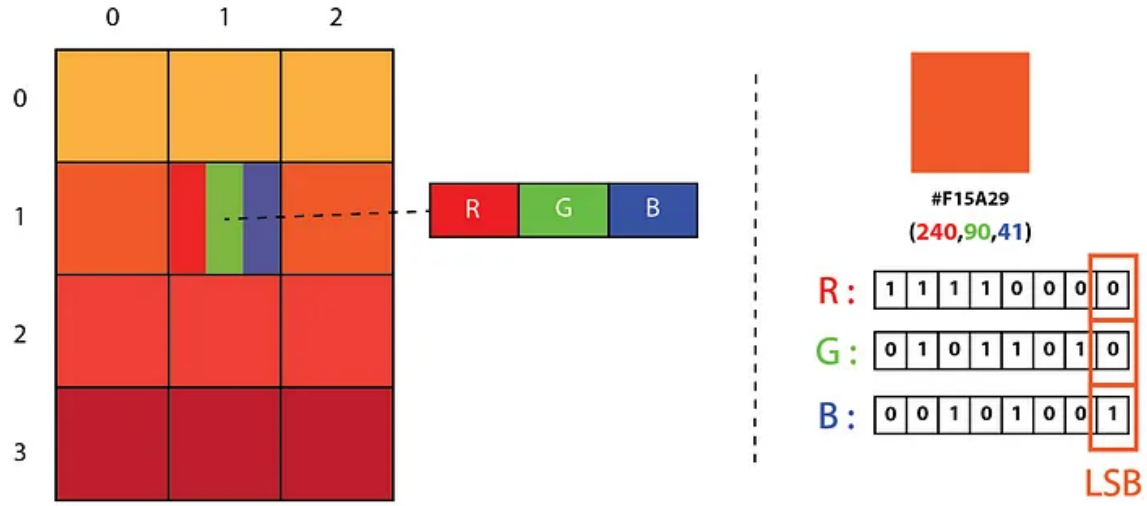
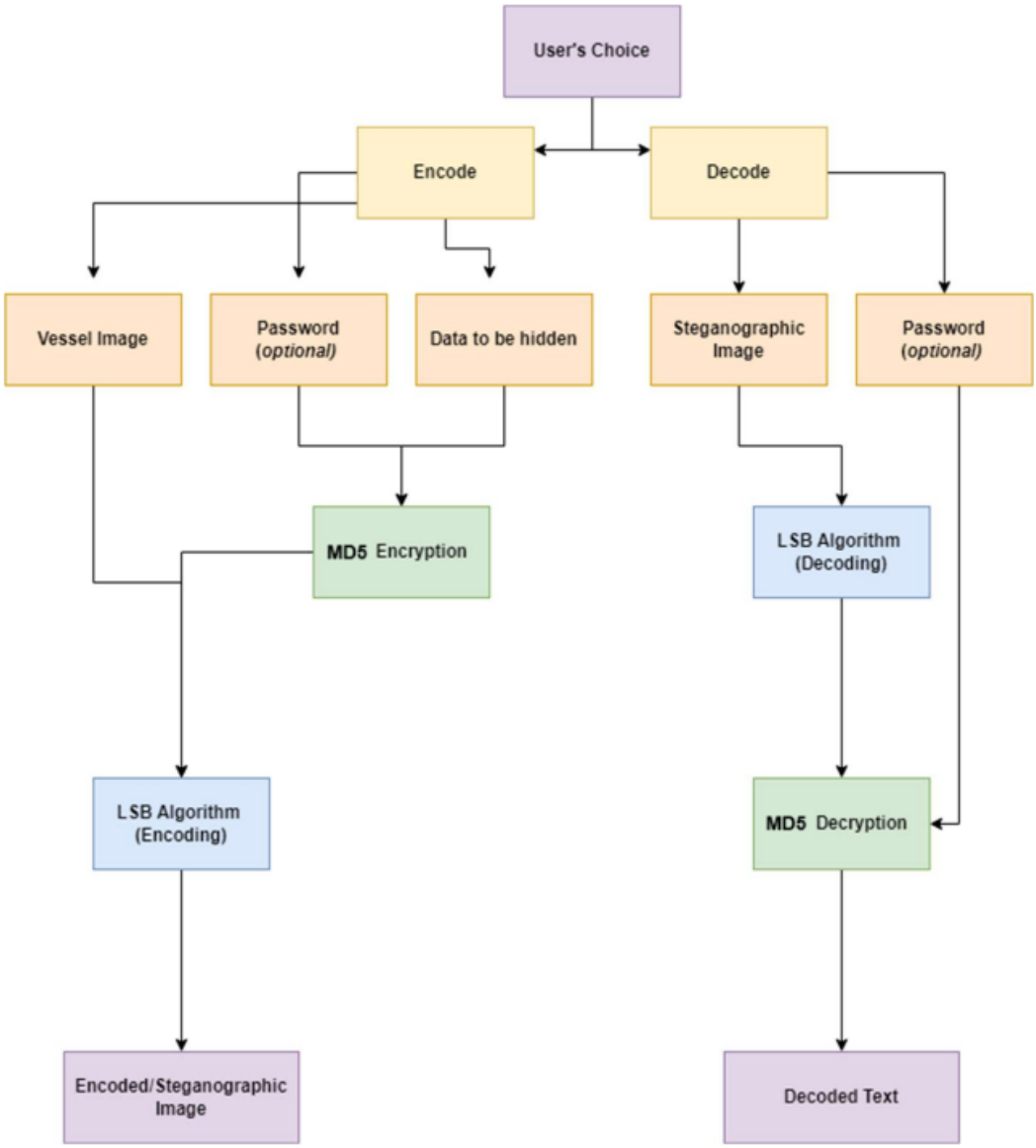


Image Steganography Using LSB Algorithm (Text-In-Image)

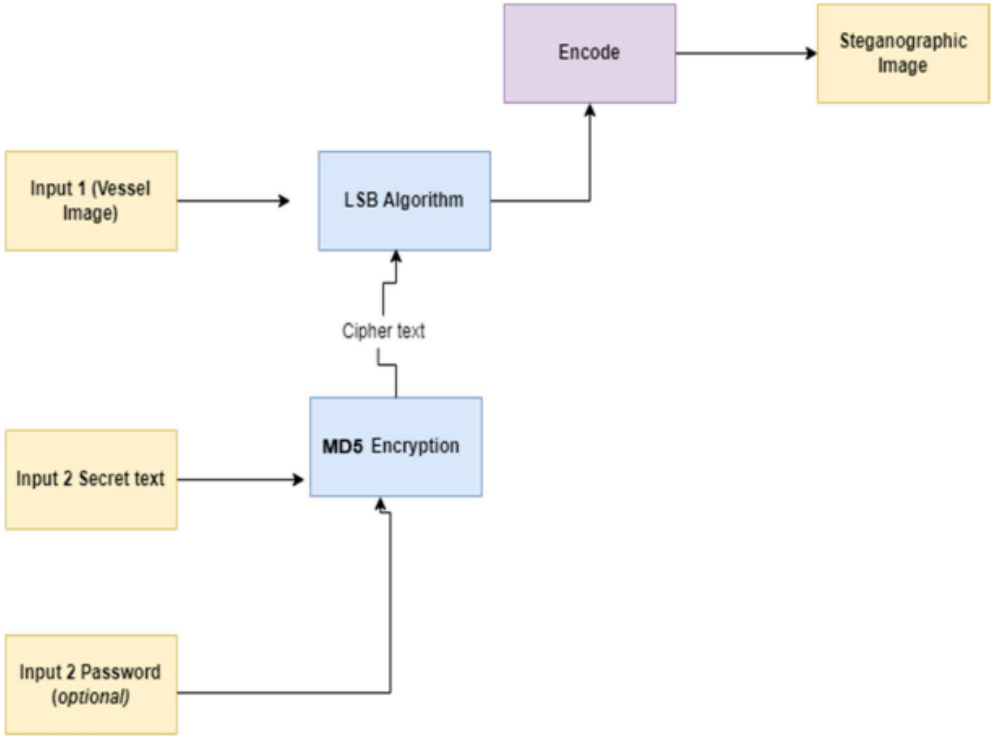
Working of Least-Significant-Bit Algorithm



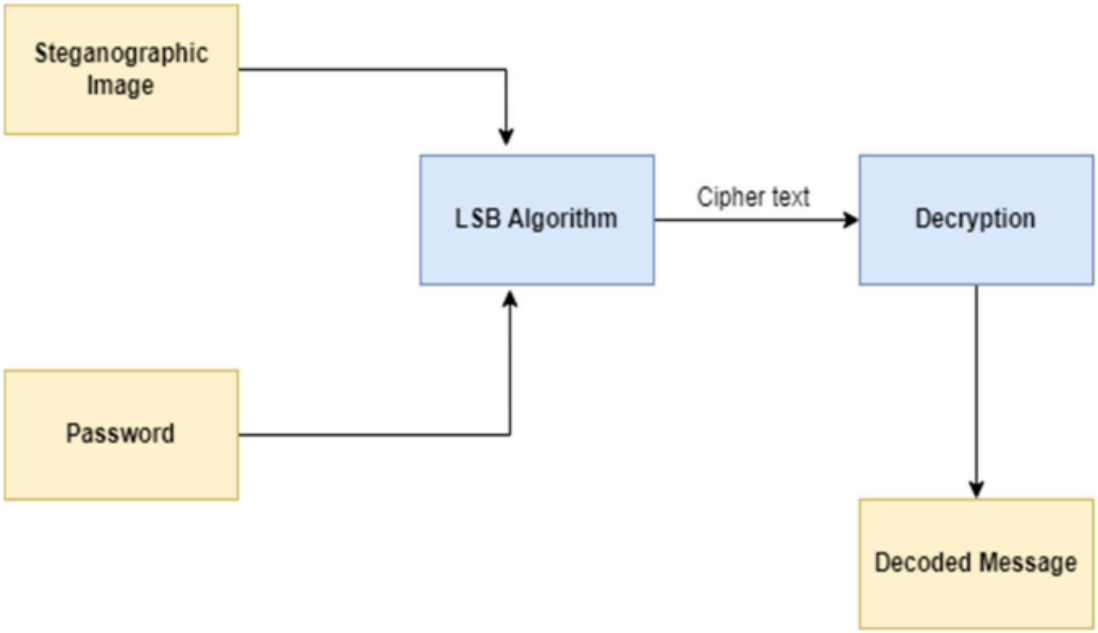
Architecture


User's Choice
Encode
Decode
Vessel Image
Password (optional)
Data to be hidden
Steganographic Image
Password (optional)
MD5 Encryption
LSB Algorithm (Decoding)
LSB Algorithm (Encoding)
MD5 Decryption
Encoded/Steganographic Image
Decoded Text

Encoding


Encode
Steganographic Image
Input 1 (Vessel Image)
LSB Algorithm
Cipher text
MD5 Encryption
Input 2 Secret text
Input 2 Password (optional)

Decoding



Implementation

NOTE: Use PNG or Lossless Images Only!

```
import cv2
from cv2 import imread,imwrite
from base64 import urlsafe_b64encode
from hashlib import md5
from cryptography.fernet import Fernet
import numpy as np
from matplotlib import pyplot as plt
from skimage.metrics import peak_signal_noise_ratio, mean_squared_error, structural_similarity



class FileError(Exception):
    pass

class DataError(Exception):
    pass

class PasswordError(Exception):
    pass


def str2bin(string):
    return ''.join((bin(ord(i))[2:]).zfill(8) for i in string)

def bin2str(string):
    return ''.join(chr(int(string[i:i+8],2)) for i in range(len(string))[::8])


def encrypt_decrypt(string,password,mode='enc'):
    _hash = md5(password.encode()).hexdigest() #get hash of password
    cipher_key = urlsafe_b64encode(_hash.encode()) #use the hash as the key of encryption
    cipher = Fernet(cipher_key) #get the cipher based on the cipher key
    if mode == 'enc':
        return cipher.encrypt(string.encode()).decode() #encrypt the data
    else:
        return cipher.decrypt(string.encode()).decode() #decrypt the data
```

```
def encode(input_filepath,text,output_filepath,password=None):
    if password != None:
        data = encrypt_decrypt(text,password,'enc') #If password is provided, encrypt the data with given password
    else:
        data = text #else do not encrypt
    data_length = bin(len(data))[2:].zfill(32) #get length of data to be encoded
    bin_data = iter(data_length + str2bin(data)) #add length of data with actual data and get the binary form of whole thi
    img = imread(input_filepath,1) #read the cover image
    if img is None:
        raise FileError("The image file '{}' is inaccessible".format(input_filepath)) #if image is not accessible, raise a
    height,width = img.shape[0],img.shape[1] #get height and width of cover image
    encoding_capacity = height*width*3 #maximum number of bits of data that the cover image can hide
    total_bits = 32+len(data)*8 #total bits in the data that needs to be hidden including 32 bits for specifying length of
    if total_bits > encoding_capacity:
        raise DataError("The data size is too big to fit in this image!") #if cover image can't hide all the data, raise D
    completed = False
    modified_bits = 0

    #Run 2 nested for loops to traverse all the pixels of the whole image in left to right, top to bottom fashion
    for i in range(height):
        for j in range(width):
            pixel = img[i,j] #get the current pixel that is being traversed
            for k in range(3): #get next 3 bits from the binary data that is to be encoded in image
                try:
                    x = next(bin_data)
                except StopIteration: #if there is no data to encode, mark the encoding process as completed
                    completed = True
                    break
                if x == '0' and pixel[k]%2==1: #if the bit to be encoded is '0' and the current LSB is '1'
                    pixel[k] -= 1 #change LSB from 1 to 0
                    modified_bits += 1 #increment the modified bits count
                elif x=='1' and pixel[k]%2==0: #if the bit to be encoded is '1' and the current LSB is '0'
                    pixel[k] += 1 #change LSB from 0 to 1
                    modified_bits += 1 #increment the modified bits count
            if completed:
                break
        if completed:
            break

    written = imwrite(output_filepath,img) #create a new image with the modified pixels
    if not written:
        raise FileError("Failed to write image '{}'".format(output_filepath))
    loss_percentage = (modified_bits/encoding_capacity)*100 #calculate how many bits of the original image are changed in
    return loss_percentage
```

```
def decode(input_filepath,password=None):
    result,extracted_bits,completed,number_of_bits = '',0,False,None
    img = imread(input_filepath) #open the image
    if img is None:
        raise FileError("The image file '{}' is inaccessible".format(input_filepath)) #if failed to open image, raise exce
    height,width = img.shape[0],img.shape[1] #get the dimensions of the image
    #Run 2 nested for loops to traverse all the pixels of the whole image in left to right, top to bottom fashion
    for i in range(height):
        for j in range(width):
            for k in img[i,j]: #for values in pixel RGB tuple
                result += str(k%2) #extract the LSB of RGB values of each pixel
                extracted_bits += 1

                if extracted_bits == 32 and number_of_bits == None: #If the first 32 bits are extracted, it is our data si
                    number_of_bits = int(result,2)*8 #number of bits to extract from the image
                    result = ''
                    extracted_bits = 0
                elif extracted_bits == number_of_bits: #if all required bits are extracted, mark the process as completed
                    completed = True
                    break
            if completed:
                break
        if completed:
            break
    if password == None: #if the data doesn't need password to be unlocked, return the string representation of binary dat
        return bin2str(result)
    else: #else, try to decrypt the data with the given password and then return the decrypted text
        try:
            return encrypt_decrypt(bin2str(result),password,'dec')
        except:
            raise PasswordError("Invalid password!") #if password did not match, raise PasswordError exception



if __name__ == "__main__":

    ch = int(input('Menu\n1.Encrypt\n2.Decrypt\n\nEnter Choice: '))
    if ch == 1:
        ip_file = input('\nEnter cover image name(path)(with extension): ')
        text = input('Enter secret data: ')
        pwd = input('Enter password: ')
        directory, filename = ip_file.rsplit('/', 1)
        new_filename = filename.replace('.', '_steg.')
        op_file = f'{directory}/{new_filename}'
        try:
            loss = encode(ip_file,text,op_file,pwd)
        except FileError as fe:
            print("Error: {}".format(fe))
        except DataError as de:
            print("Error: {}".format(de))
        else:
            print('Encoded Successfully!'.format(loss))
    elif ch == 2:
        ip_file = input('Enter image path: ')
        pwd = input('Enter password: ')
        try:
            data = decode(ip_file,pwd)
        except FileError as fe:
            print("Error: {}".format(fe))
        except PasswordError as pe:
            print('Error: {}'.format(pe))
        else:
            print('Decrypted data:',data)
    else:
        print('Wrong Choice!')
```

```
 Menu
 1.Encrypt
 2.Decrypt

 Enter Choice: 1

 Enter cover image name(path)(with extension): /content/super car.jpg
 Enter secret data: abc
```

```
 Enter password: abc
 Encoded Successfully!
```

```
if __name__ == "__main__":

    ch = int(input('Menu\n1.Encrypt\n2.Decrypt\n\nEnter Choice: '))
    if ch == 1:
        ip_file = input('\nEnter cover image name(path)(with extension): ')
        text = input('Enter secret data: ')
        pwd = input('Enter password: ')
        directory, filename = ip_file.rsplit('/', 1)
        new_filename = filename.replace('.', '_steg.')
        op_file = f'{directory}/{new_filename}'
        try:
            loss = encode(ip_file,text,op_file,pwd)
        except FileError as fe:
            print("Error: {}".format(fe))
        except DataError as de:
            print("Error: {}".format(de))
        else:
            print('Encoded Successfully!\nImage Data Loss = {:.5f}%'.format(loss))
    elif ch == 2:
        ip_file = input('Enter image path: ')
        pwd = input('Enter password: ')
        try:
            data = decode(ip_file,pwd)
        except FileError as fe:
            print("Error: {}".format(fe))
        except PasswordError as pe:
            print('Error: {}'.format(pe))
        else:
            print('Decrypted data:',data)
    else:
        print('Wrong Choice!')
```

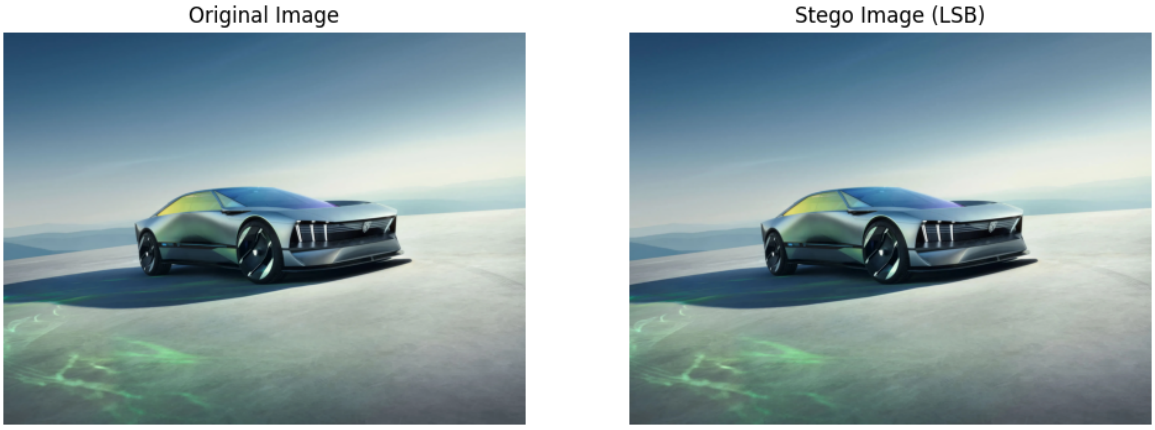
```
 Menu
 1.Encrypt
 2.Decrypt

 Enter Choice: 2
 Enter image path: /content/super car_steg.jpg
 Enter password: 123
 Error: Invalid password!
```

```
if __name__ == "__main__":

    ch = int(input('Menu\n1.Encrypt\n2.Decrypt\n\nEnter Choice: '))
    if ch == 1:
        ip_file = input('\nEnter cover image name(path)(with extension): ')
        text = input('Enter secret data: ')
        pwd = input('Enter password: ')
        directory, filename = ip_file.rsplit('/', 1)
        new_filename = filename.replace('.', '_steg.')
        op_file = f'{directory}/{new_filename}'
        try:
            loss = encode(ip_file,text,op_file,pwd)
        except FileError as fe:
            print("Error: {}".format(fe))
        except DataError as de:
            print("Error: {}".format(de))
        else:
            print('Encoded Successfully!\nImage Data Loss = {:.5f}%'.format(loss))
    elif ch == 2:
        ip_file = input('Enter image path: ')
        pwd = input('Enter password: ')
        try:
            data = decode(ip_file,pwd)
        except FileError as fe:
            print("Error: {}".format(fe))
        except PasswordError as pe:
            print('Error: {}'.format(pe))
        else:
            print('Decrypted data:',data)
```

```
            print('Decrypted data: ',data)
    else:
        print('Wrong Choice!')

     Menu
     1.Encrypt
     2.Decrypt

     Enter Choice: 2
     Enter image path: /content/super car_steg.jpg
     Enter password: abc
     Error: Invalid password!
```

Visual Representations

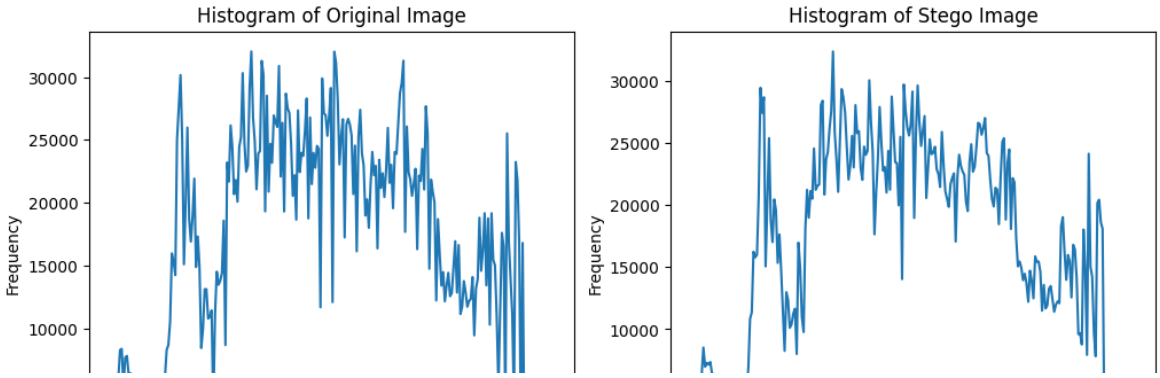
```
ip_image = cv2.cvtColor(cv2.imread(ip_file), cv2.COLOR_BGR2RGB)
stego_image = cv2.cvtColor(cv2.imread(op_file), cv2.COLOR_BGR2RGB)

plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.imshow(ip_image)
plt.title('Original Image')
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(stego_image)
plt.title('Stego Image (LSB)')
plt.axis('off')

plt.show()
```



As seen above, No visual differences in both images.

Histogram Representations

```
hist_original = cv2.calcHist([ip_image], [0], None, [256], [0, 256])
hist_stego = cv2.calcHist([stego_image], [0], None, [256], [0, 256])

plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(hist_original)
plt.title('Histogram of Original Image')
plt.xlabel('Pixel Value')
plt.ylabel('Frequency')

plt.subplot(1, 2, 2)
plt.plot(hist_stego)
plt.title('Histogram of Stego Image')
plt.xlabel('Pixel Value')
plt.ylabel('Frequency')

plt.show()
```



As seen, there is no noticeable difference in histograms of both images.

Comparison of Metrics

```
def calculate_metrics(original, steg):
    psnr_value = peak_signal_noise_ratio(original, steg)
    mse_value = mean_squared_error(original, steg)
    ssim_value, _ = structural_similarity(original, steg, full=True)

    return psnr_value, mse_value, ssim_value


original_image = cv2.cvtColor(ip_image, cv2.COLOR_BGR2GRAY)
steg_image = cv2.cvtColor(stego_image, cv2.COLOR_BGR2GRAY)
```