**INFORMATION SECURITY PROJECT**

**Network Security**

**Submitted By:**

**Aditri Srivastava (9919103092)**

**Aryan Kumar (9919103093)**

**Shazia Salim (9919103099)**

**Semester-5th**

**Under Supervision Of:**

**BANSIDHAR JOSHI**

****

**Department of CSE/IT**

**Jaypee Institute of Information Technology University, Noida**

**Steganography**

Steganography is the process of hiding a secret message within a larger one in such a way that someone cannot know the presence or contents of the hidden message. The purpose of Steganography is to maintain secret communication between two parties. Unlike cryptography, which conceals the contents of a secret message, steganography conceals the very fact that a message is communicated. Although steganography differs from cryptography, there are many analogies between the two, and some authors classify steganography as a form of cryptography since hidden communication is a type of secret message. The secret can be in the form of a text or in the form of an image (i.e., Hiding Text inside an Image or hiding Image inside an Image)

**Abstract of the project**

Steganography is the art of hiding the fact that communication is taking place, by hiding information in other information. Many different carrier file formats can be used, but digital images are the most popular because of their frequency on the internet. For hiding secret information in images, there exists a large variety of steganography techniques, some are more complex than others and all of them have respective strong and weak points. Different applications may require absolute invisibility of the secret information, while others require a large secret message to be hidden. This project report intends to give an overview of image steganography, its uses and techniques. It also attempts to identify the requirements of a good steganography algorithm and briefly reflects on which steganographic techniques are more suitable for which applications.

**Scope of the project**

● **Functional requirements**

1. Open CV

2. Least Significant Bit Steganography

3. Python Imaging Library

● **Non-functional requirements**

1. Jupyter Notebook

2. Collab

3. VS Code

**Library description**

OpenCV-Python is a library of Python bindings designed to solve computer vision problems. OpenCV-Python makes use of NumPy, which is a highly optimized library for numerical operations with a MATLAB-style syntax. All the OpenCV array structures are converted to and from NumPy arrays. This also makes it easier to integrate with other libraries that use NumPy such as SciPy and Matplotlib.

The Python Imaging Library adds image processing capabilities to your Python interpreter. This library provides extensive file format support, an efficient internal representation, and fairly powerful image processing capabilities. The core image library is designed for fast access to data stored in a few basic pixel formats. It should provide a solid foundation for a general image processing tool.

**Design of the project**

The project has two python files one for hiding text in image and other for hiding image in Image.

● Hiding Text in Image

We are hiding the text in image. For encryption we take a key and message to hide and take XOR of it in image.

For decryption again we ask for the key and if the key is matched, we perform XOR operation and find the message.

● Hiding Image in Image

○ We take paths of the image to hide and image to hide in after that we read the images keeping number of bits as 6.

○ Encoded function is called that encodes the image in image.

■ We take the n significant bits from the image to be hidden and n least significant bits from the image to hide in.

■ adding these two values and append in it data array

■ After that, we convert data to image and get encrypted image. ○ To decode image, Decoded Function is called and decodes the images in the reverse manner as done in encode function that is removing least significant bits from the encoded image.

**Implementation details**

**Program Code:**

**Text to Image**

import cv2

import string

import os

d={}

c={}

for i in range(255):

d[chr(i)]=i

c[i]=chr(i)

#print(c)

x=cv2.imread("lion.jpg")

i=x.shape[0]

j=x.shape[1]

print(i,j)

key=input("Enter key to edit(Security Key) : ")

text=input("Enter text to hide : ")

kl=0

tln=len(text)

z=0 #decides plane

n=0 #number of row

m=0 #number of column

l=len(text)

for i in range(l):

x[n,m,z]=d[text[i]]^d[key[kl]]

n=n+1

m=m+1

z=(z+1)%3 #this is for every value of z , remainder will be between 0,1,2 . i.e G,R,B plane will be set automatically.

#whatever be the value of z , z=(z+1)%3 will always between 0,1,2 . The same concept is used for random number in dice and card games.

kl=(kl+1)%len(key)

cv2.imwrite("lion.jpg",x)

os.startfile("lion.jpg")

print("Data Hiding in Image completed successfully.") #x=cv2.imread(“encrypted\_img.jpg")

kl=0

tln=len(text)

z=0 #decides plane

n=0 #number of row

m=0 #number of column

ch = int(input("\nEnter 1 to extract data from Image : "))

if ch == 1:

key1=input("\n\nRe enter key to extract text : ") decrypt=""

if key == key1 :

for i in range(l):

decrypt+=c[x[n,m,z]^d[key[kl]]]

n=n+1

m=m+1

z=(z+1)%3

kl=(kl+1)%len(key)

print("Encrypted text was : ",decrypt) else:

print("Key doesn't matched.")

else:

print("Thank you. EXITING.")

**Image in Image**

from PIL import Image

MAX\_COLOR\_VALUE = 256

MAX\_BIT\_VALUE = 8

def make\_image(data, resolution):

image = Image.new("RGB", resolution) image.putdata(data)

return image

def remove\_n\_least\_significant\_bits(value, n): value = value >> n

return value << n

def get\_n\_least\_significant\_bits(value, n):

value = value << MAX\_BIT\_VALUE - n

value = value % MAX\_COLOR\_VALUE

return value >> MAX\_BIT\_VALUE - n

def get\_n\_most\_significant\_bits(value, n): return value >> MAX\_BIT\_VALUE - n

def shit\_n\_bits\_to\_8(value, n):

return value << MAX\_BIT\_VALUE - n

def encode(image\_to\_hide, image\_to\_hide\_in, n\_bits): width, height = image\_to\_hide.size

hide\_image = image\_to\_hide.load()

hide\_in\_image = image\_to\_hide\_in.load() data = []

for y in range(height):

for x in range(width):

r\_hide, g\_hide, b\_hide = hide\_image[x,y]

r\_hide = get\_n\_most\_significant\_bits(r\_hide, n\_bits) g\_hide = get\_n\_most\_significant\_bits(g\_hide, n\_bits) b\_hide = get\_n\_most\_significant\_bits(b\_hide, n\_bits)

# remove lest n sig bits

r\_hide\_in, g\_hide\_in, b\_hide\_in = hide\_in\_image[x,y]

r\_hide\_in = remove\_n\_least\_significant\_bits(r\_hide\_in, n\_bits) g\_hide\_in = remove\_n\_least\_significant\_bits(g\_hide\_in, n\_bits) b\_hide\_in = remove\_n\_least\_significant\_bits(b\_hide\_in, n\_bits)

data.append((r\_hide + r\_hide\_in,

g\_hide + g\_hide\_in,

b\_hide + b\_hide\_in))

return make\_image(data, image\_to\_hide.size)

def decode(image\_to\_decode, n\_bits):

width, height = image\_to\_decode.size

encoded\_image = image\_to\_decode.load()

data = []

for y in range(height):

for x in range(width):

r\_encoded, g\_encoded, b\_encoded = encoded\_image[x,y]

r\_encoded = get\_n\_least\_significant\_bits(r\_encoded, n\_bits) g\_encoded = get\_n\_least\_significant\_bits(g\_encoded, n\_bits) b\_encoded = get\_n\_least\_significant\_bits(b\_encoded, n\_bits)

r\_encoded = shit\_n\_bits\_to\_8(r\_encoded, n\_bits) g\_encoded = shit\_n\_bits\_to\_8(g\_encoded, n\_bits) b\_encoded = shit\_n\_bits\_to\_8(b\_encoded, n\_bits)

data.append((r\_encoded, g\_encoded, b\_encoded)) return make\_image(data, image\_to\_decode.size)

if "\_main\_":

image\_to\_hide\_path = "./c.jpeg"

image\_to\_hide\_in\_path = "./d.jpeg"

encoded\_image\_path = "./encoded.jpeg"

decoded\_image\_path = "./decoded.jpeg"

n\_bits = 6

image\_to\_hide = Image.open(image\_to\_hide\_path)

image\_to\_hide\_in = Image.open(image\_to\_hide\_in\_path)

print(image\_to\_hide.size)

print(image\_to\_hide\_in.size)

# image\_to\_hide.resize((259,194))

# image\_to\_hide\_in = image\_to\_hide\_in.resize((259,194), Image.ANTIALIAS) # print(image\_to\_hide.size)

# print(image\_to\_hide\_in.size)

# # image\_to\_hide\_in.resize((40,40))

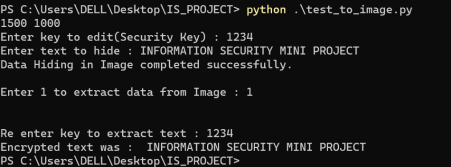
encode(image\_to\_hide, image\_to\_hide\_in, n\_bits).save(encoded\_image\_path)

image\_to\_decode = Image.open(encoded\_image\_path)

decode(image\_to\_decode, n\_bits).save(decoded\_image\_path)

**Screenshots and explanation**

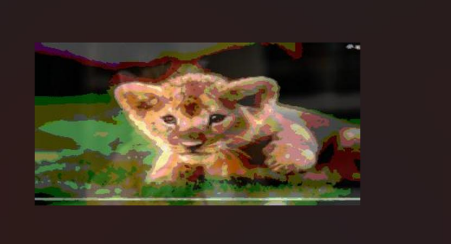
**Text to Image**

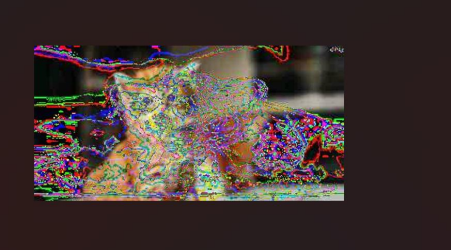
**Encoded Image:**

****

**Image in Image**

**Encrypted Image:**

**Decrypted Image:**

****

**References**

● https://pypi.org/project/Pillow/

● https://pypi.org/project/opencv-python/

● https://en.wikipedia.org/wiki/Steganography

● https://towardsdatascience.com/steganography-hiding-an-image-inside another-77ca66b2acb1

● https://www.researchgate.net/publication/314116270\_Image\_Steganography