ELL205 Project Report

Image Message Encryption and Decryption

Aditya Singal 2021EE31046

Srijan Singh 2021EE10675

Objectives-

1. To understand and give an outline of the proposed method of Encrypting and Decrypting Images using XOR operation.
2. Write Python Code for this Image Encryption and Decryption algorithm and create a working model.

Abstract

Encryption is one of the best techniques to ensure security during transmission of image and video content over the internet and the web. Images are used in various areas so its security is of great concern nowadays. To focus on the security aspect, in this paper we are looking at a novel method of image encryption and decryption using Chaos, Logistic Maps and Bit Wise XOR operation.

Introduction

Encryption is the process of encoding messages so that eavesdroppers or hackers cannot read them, but only authorized entities can.

Image encryption approaches attempt to turn a picture into another that is difficult to interpret. Image decryption, on the other hand, recovers the original image from the encrypted one.

There are various encryption Algorithms, most of which involve the technique of creating a key using which, we convert the information to an incomprehensible form that is only retrieved through that key.

Based on the key, the encryption algorithm can be classified into two categories. They are-

- The **symmetric** key encryption- This algorithm uses the same key for both encryption and decryption.
- The **asymmetric** key encryption- This algorithm uses different keys for encryption and decryption.

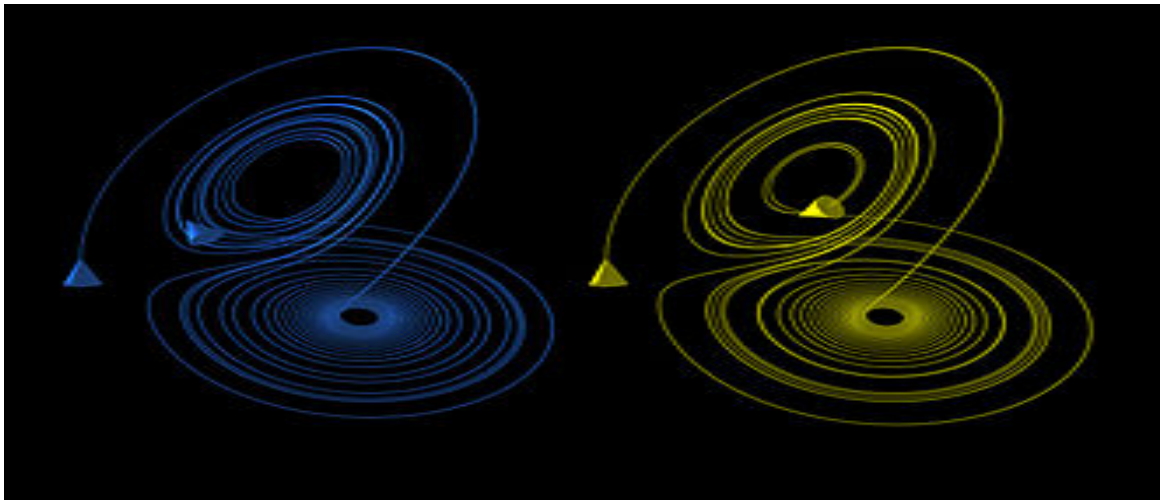
The asymmetric key algorithm has very higher computational costs than Symmetric key encryption algorithms, which have a comparatively lower cost.

Prerequisite Knowledge

Chaos Theory

Chaos theory is a scientific discipline and area of mathematics focusing on the underlying patterns and deterministic principles of **extremely sensitive dynamical systems.**
The Butterfly Effect is a well-known example.
Small changes in initial circumstances, such as those caused by **measurement mistakes or rounding errors in numerical** calculation, can produce significantly differing results for such dynamical systems, making a long-term prediction of their behavior difficult.

The Lorenz attractor displays chaotic behavior. These two plots depict the dependence on slight changes in the initial conditions within the of phase space occupied by the attractor.

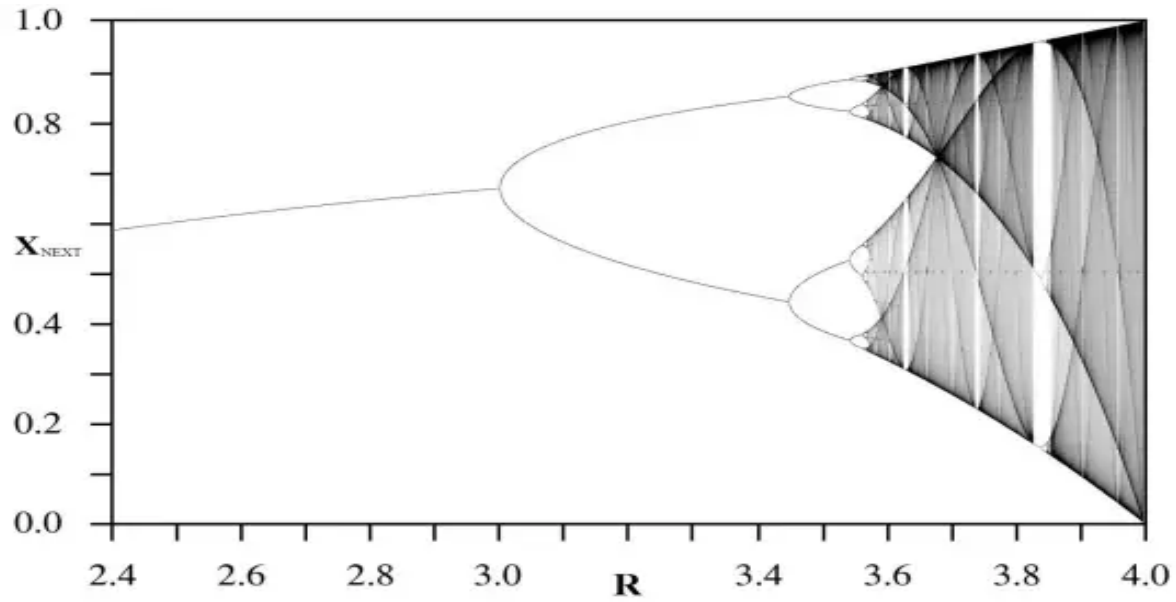


We have used this concept to make our encryption key such that unless the absolutely accurate value of the key is not known, even the closest approximation cannot be used to decrypt the image. Even a small change in the initial value of the key would result in a completely different encrypted image, as observed in many dynamic systems.

Logistics Map

The logistic map is a degree 2 polynomial mapping (equivalently, recurrence relation) that is sometimes referenced as a quintessential illustration of how complex, chaotic behavior may evolve from extremely simple non-linear dynamical equations.

$x_{n+1}=rx_n(1+x_n)$



Following Image: Is the bifurcation diagram showing the change in x_next as r is increased. We take input x and r in our code to generate keys. For values of r greater than 3.6, we get complete chaotic behavior.

Algorithm

1. We can think of an image as just an nXm matrix where n is the height of the image in pixels and m is the width of the image in pixels. We will call this matrix the image matrix. G_{nXm}

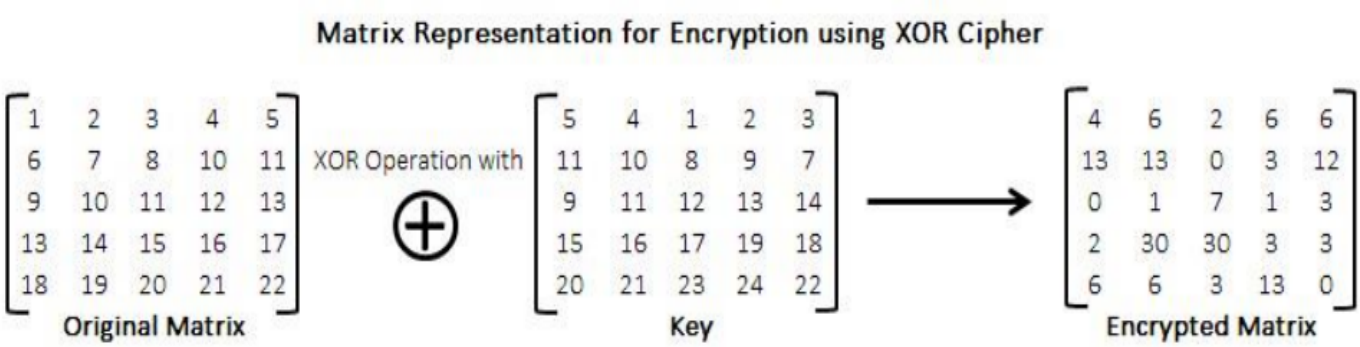
   Code Fragment:
   ```
   img = mplimg.imread('sample.bmp')
   ```

2. Using our knowledge of Chaos Theory and Logistic Maps, we essentially generate a key matrix using suitable values of r and x which are our input parameters to the function gen_keys. This generates an *nxm* keys matrix K_{nXm}.

Code fragment:

```
def gen_keys(x,r,size):
     keys=[]
     for _ in range(size):
         x=r*x*(1-x) #logistic map
keys.append(int((x*pow(10,16))%256))#key=(x*10^16)%256
    return keys
keys = gen_keys(0.011,3.95,height*width)
```

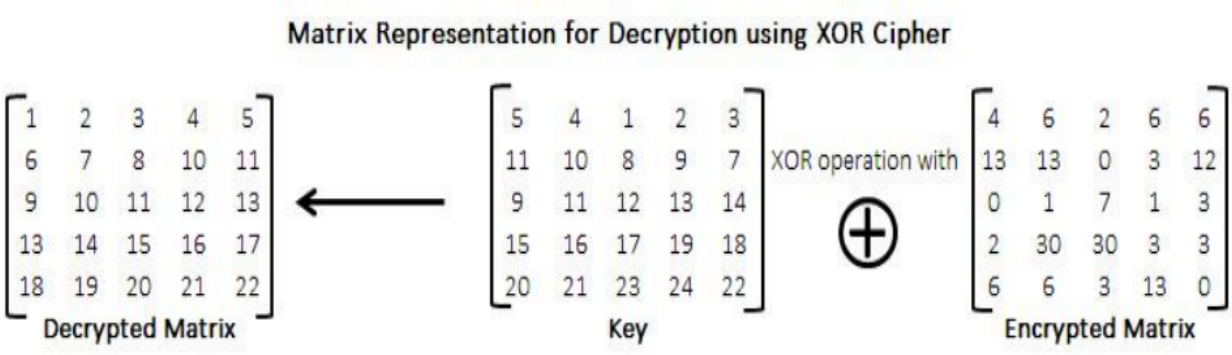
3. **Encryption:**
    a. We create a new null nxm matrix which is called the encrypted image matrix E_{nxm}.
    b. We assign the entries of this encrypted matrix E_{ij} by using XOR operation between G_{ij} and K_{ij}
    c. This creates the encrypted image.



Code Fragment for Encryption:

```
z = 0
enimg = np.zeros(shape = [height,width,4], dtype =
np.uint8)
for i in range(height):
    for j in range(width):
        enimg[i,j] = img[i,j]^keys[z]
        z+=1
pl0t.imshow(enimg)
pl0t.show()
pl0t.imsave('enc_img.bmp',enimg)
```

4. **Decryption:**
    a. We create a new null nXm matrix which is called the decrypted image matrix D_{nXm}.
    b. We assign the entries of this decrypted matrix D_{ij} by using XOR operation between E_{ij} and K_{ij}
    c. This creates the decrypted image.



Code Fragment:

```
z = 0
decimg = np.zeros(shape = [height,width,4], dtype =
np.uint8)
for i in range(height):
    for j in range(width):
        decimg[i,j] = enimg[i,j]^keys[z]
        z+=1
plt.imshow(decimg)
plt.show()
plt.imsave('dec_img.bmp',decimg)
```

6. Plotting Histograms:

- We first read the image from the folder.
- Then split the image into its component RBG using the Opencv module

Code fragment for the following:

```
import numpy as np
import cv2 as open
from matplotlib import pyplot as pl0t
image = open.imread("fame.jpeg")
blue, green, red = open.split(image)
open.imshow("image", image)
```

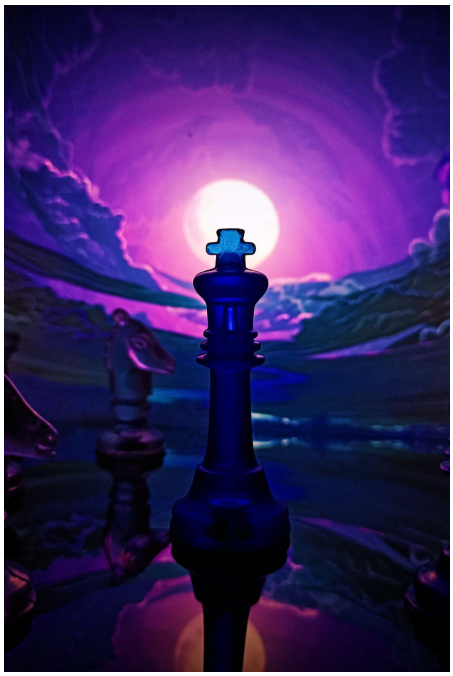
- Following we use the pl0t.hist() function from matplotlib module to plot the intensity of each component.
- Then we used calcHist() function to trace out and plot the histogram.

Code fragment for the following:

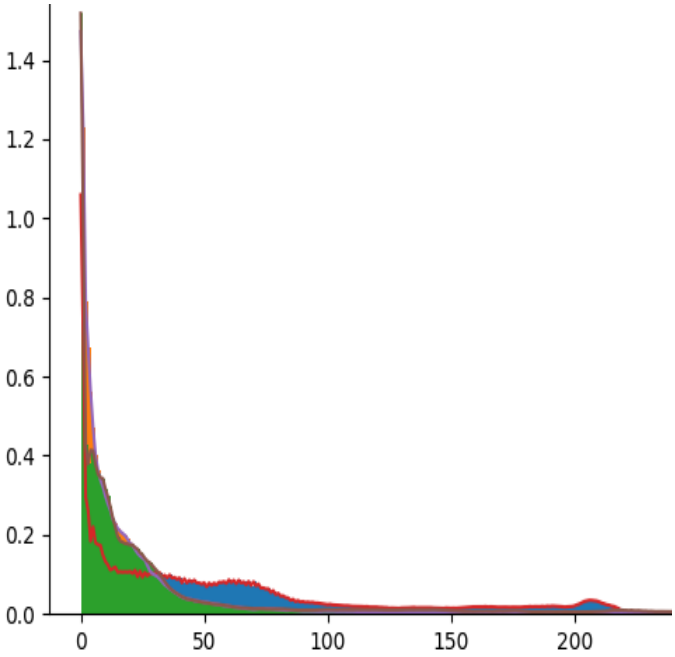
```
pl0t.hist(blue.ravel(), 256, [0, 256])
pl0t.hist(green.ravel(), 256, [0, 256])
pl0t.hist(red.ravel(), 256, [0, 256])

hist_2 = open.calcHist([image], [2], None, [256], [0, 256])
hist_1 = open.calcHist([image], [1], None, [256], [0, 256])
```

```
hist = open.calcHist([image], [0], None, [256], [0, 256])
#exposure histogram
pl0t.plot(hist)
pl0t.plot(hist1)
pl0t.plot(hist2)
pl0t.show()
```



Original Image



Histogram obtained

Complete Code / Working Model

```
import matplotlib.pyplot as pl0t
import matplotlib.image as mplimg
import numpy as np
```

```
def gen_keys(x,r,size):
    keys=[]
    for _ in range(size):
        x=r*x*(1-x) #logistic map
        keys.append(int((x*
pow(10,16))%256))#key=(x*10^16)%256
    return keys

img = mplimg.imread('sample.bmp')
pl0t.imshow(img)
pl0t.show()
height = img.shape[0]
width = img.shape[1]
keys = gen_keys(0.011,3.95,height*width)
z = 0
en_img = np.zeros(shape = [height,width,4], dtype =
np.uint8)

for i in range(height):

    for j in range(width):

        en_img[i,j] = img[i,j]^keys[z]
        z+=1

pl0t.imshow(en_img)
```

```
pl0t.show()
pl0t.imsave('enc_img.bmp',en_img)
```

```
z = 0
dec_img = np.zeros(shape = [height,width,4], dtype =
np.uint8)
for i in range(height):
    for j in range(width):
        dec_img[i,j] = en_img[i,j]^keys[z]
        z+=1

pl0t.imshow(dec_img)
pl0t.show()
pl0t.imsave('dec_img.bmp',decimg)
```

```
import cv2 as open
image = open.imread("fame.jpeg")
blue, green, red = open.split(image)
open.imshow("image", image)
```

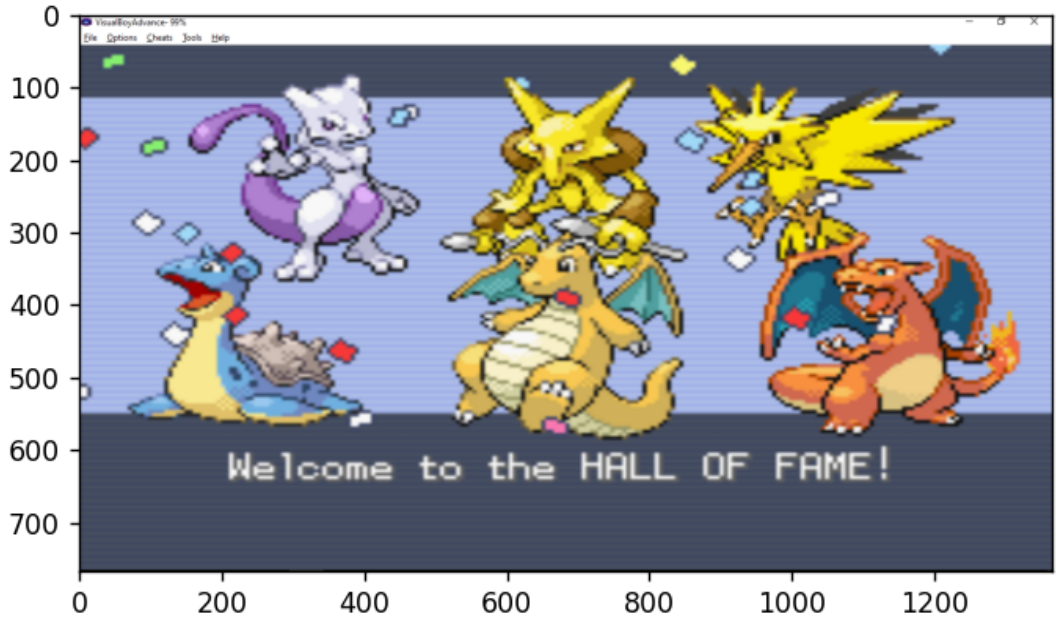
```
pl0t.hist(blue.ravel(), 256, [0, 256])
pl0t.hist(green.ravel(), 256, [0, 256])
pl0t.hist(red.ravel(), 256, [0, 256])

hist_1 = open.calcHist([image], [1], None, [256], [0, 256])
hist_2 = open.calcHist([image], [2], None, [256], [0, 256])
```

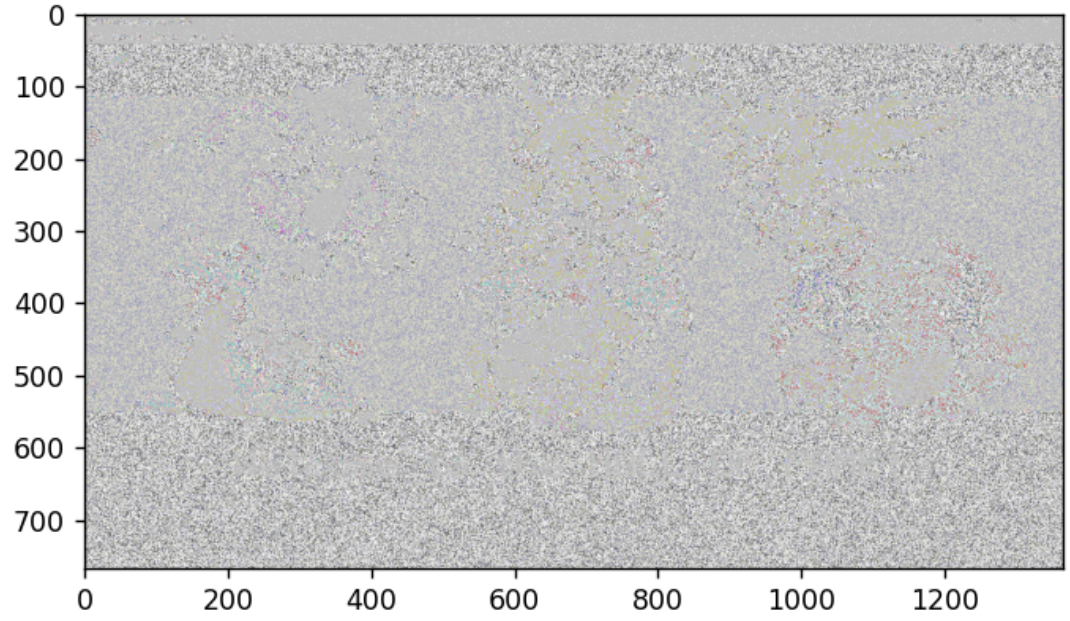
```
hist = open.calcHist([image], [0], None, [256], [0, 256])
#exposure histogram
pl0t.plot(hist)
pl0t.plot(hist_1)
pl0t.plot(hist_2)
pl0t.show()
```

RESULTS

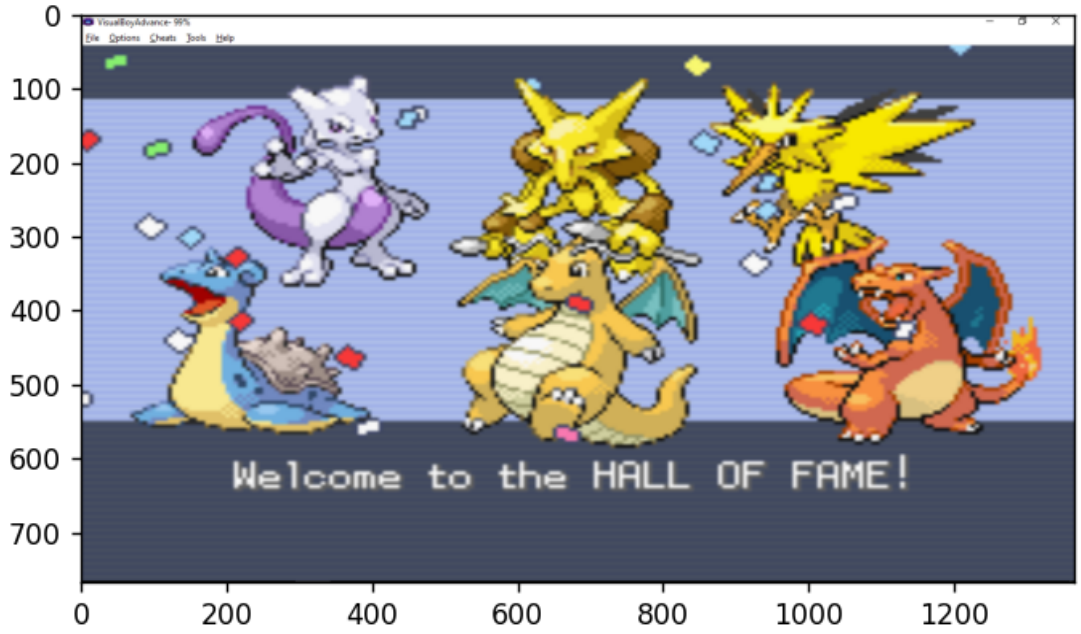
Original Image



Encrypted Image



Decrypted Image

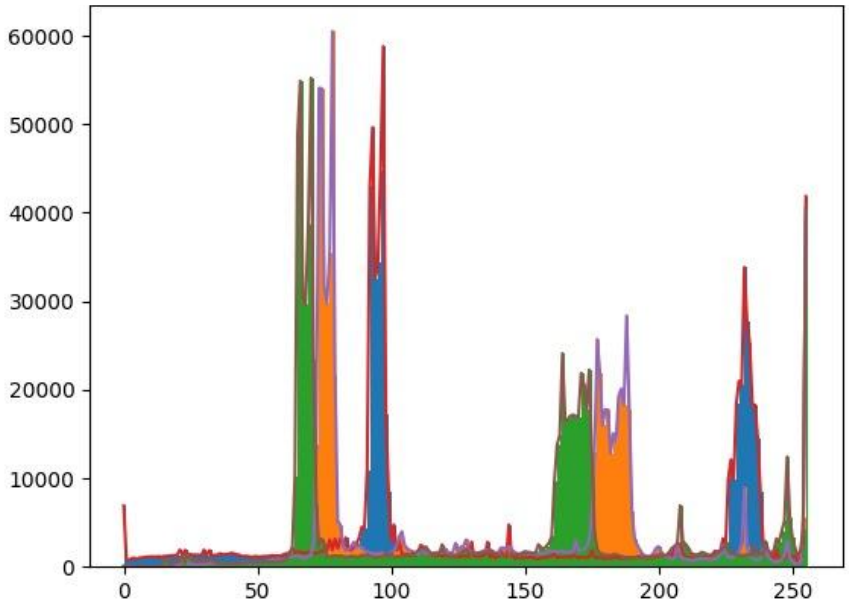


Histograms

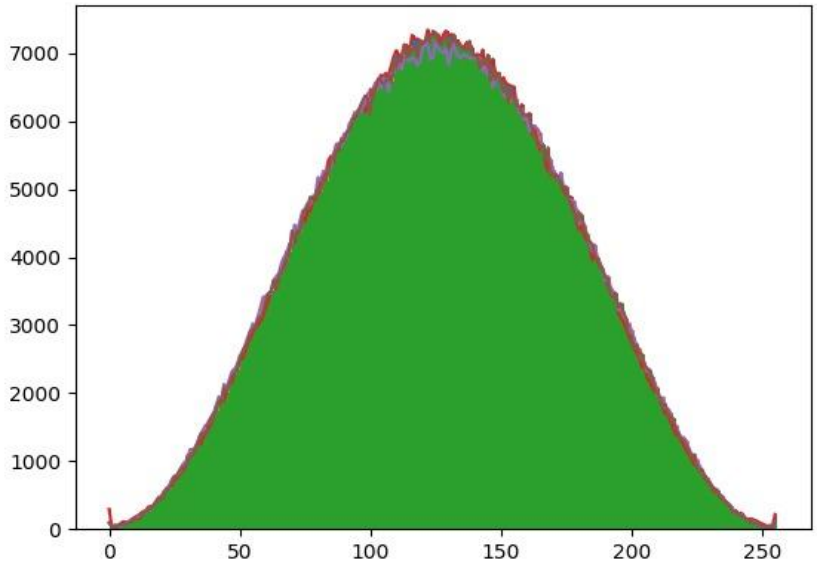
The Histogram gives an estimation of how the pixels are distributed. The histogram thus gives an idea of the randomness of the distribution of the pixel values in the image. It is very difficult to decrypt an image if the randomness of the encrypted image is high. We can see from

the following histograms that Encryption using bitwise XOR operator is very useful in securing images.

Histogram of Original Image



Histogram of the Encrypted Image



REFERENCES

1. https://www.ijert.org/research/image-encryption-using-random-scrambling-and-xor-operation-IJERTV2IS3297.pdf
2. https://en.wikipedia.org/wiki/Logistic_map
3. https://trp.org.in/wp-content/uploads/2018/05/AJCST-Vol.7-No.1-January-June-2018-pp.-55-60.pdf
4. https://www.youtube.com/watch?v=fDek6cYijxI&t=620s
5. https://www.youtube.com/watch?v=ovJcsL7vyrk&t=1005s