Configuration Manual

MSc Research Project
MSc In Cybersecurity

Mayur Gaikwad
Student ID: x22183655

School of Computing
National College of Ireland

Supervisor: Prof. Joel Aleburu

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Mayur Dattajirao Gaikwad		
Student ID:	X22183655		
Programme:	MSc in Cybersecurity	**Year:**	2023-2024
Module:	MSc Research Project/Internship		
Lecturer:	Prof. Joel Aleburu		
Submission Due Date:	12/08/2024		
Project Title:	Enhanced IoT Image Encryption: A Hybrid Approach Using Duffing and Henon Chaotic Systems		
Word Count: 1329	**Page Count:** 20		

Signature: Mayur D. Gaikwad

Date: 12/08/2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Mayur Gaikwad
X22183655

1 Overview

The project titled “The image encryption chaotic mapping” involves using of the chaotic mapping to encrypt images. This blended learning method comprises of ideas from the three chaotic systems in the ipynb file (google colab).

1. **tinker-henon:** Combines both the Tinker and Henon chaotic maps
2. **tinker-duffing:** Combines both the Duffing and Tinker chaotic maps
3. **duffing-henon:** Combines both the Duffing and Henon chaotic maps.

2 Hardware Configuration

A. Operating system: Windows >= 7

B. Processor: Intel >=i2

C. System Compatibility: 64-bit

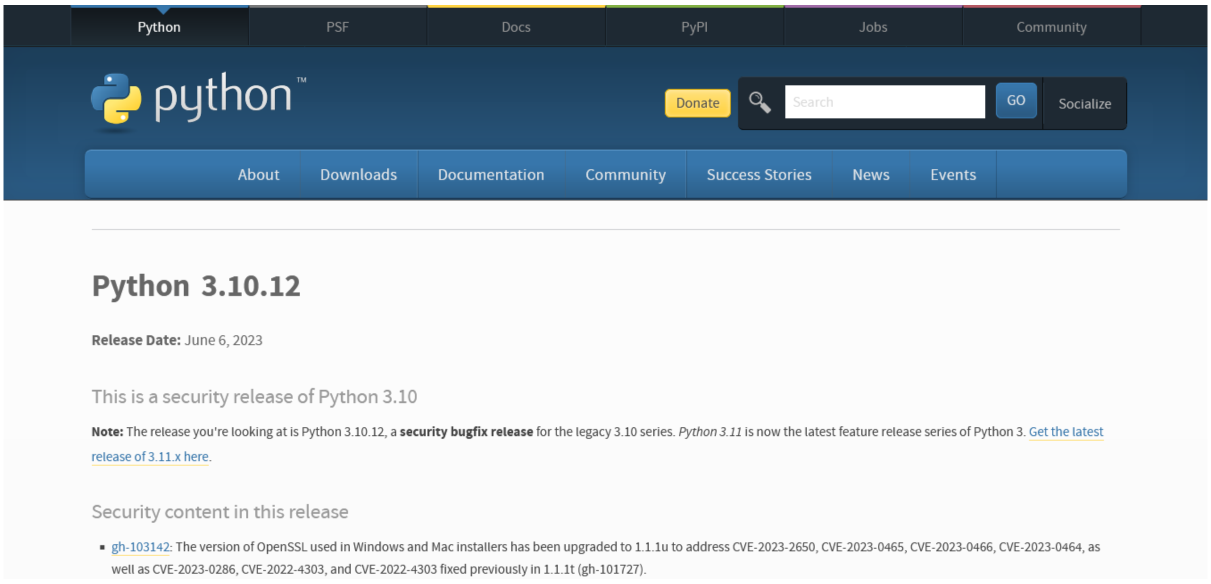
D. Hard Disk: 500 GB

E. RAM: >=4 GB

3 Software Configuration

3.1 Python 3.10.12

Python is classified as a general-purpose language which has fairly easy writing procedures and also aims at minimizing the amount of writing required for the code; hence Python is suitable for first-time learners as well as high-level users. Regarding the third benefit, it is devoted to the simple syntax of the language, which allows the programmer to introduce some notion with the help of fewer lines of code compared to other programming languages besides increasing productivity, it reduces the risk of the inclusion of mistakes. Python can therefore be regarded as an interpreted, high level language which supports procedural, object oriented/functional paradigms and is distinguished by an ability to encompass a vast quantity of packages/libraries for web development, data/machine learning/AI and sundry others. They have made Python to be adopted in severally areas including software development and scientific calculations (Python, 2019).



Python 3.10.12

Release Date: June 6, 2023

This is a security release of Python 3.10

Note: The release you're looking at is Python 3.10.12, a **security bugfix release** for the legacy 3.10 series. *Python 3.11* is now the latest feature release series of Python 3. Get the latest release of 3.11.x here.

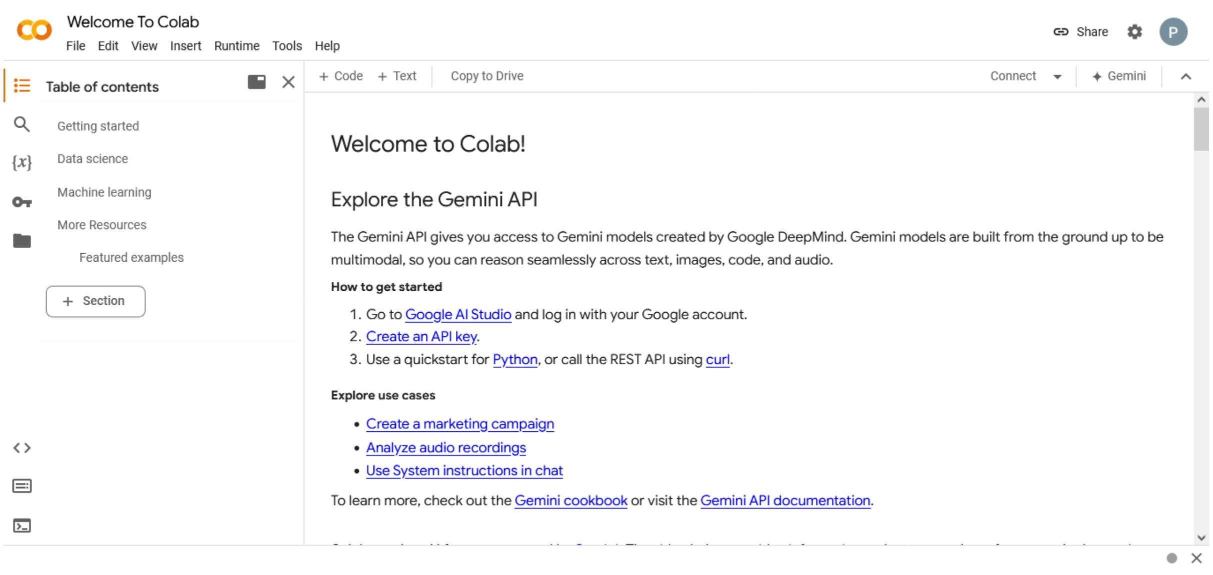
Security content in this release

- gh-103142: The version of OpenSSL used in Windows and Mac installers has been upgraded to 1.1.1u to address CVE-2023-2650, CVE-2023-0465, CVE-2023-0466, CVE-2023-0464, as well as CVE-2023-0286, CVE-2022-4303, and CVE-2022-4303 fixed previously in 1.1.1t (gh-101727).

Figure 3.1: Python Code

3.2 Google Colab

Google Colab is a free product of Google through which writing and running code in Python is possible in notebook form. It integrates with Google Drive for files' handling and it supports multiple machine learning frameworks, including TensorFlow and PyTorch. Colab also has the provision of GPU and TPU boost which makes it quite relevant for data scientist, researcher, and developer especially for machine learning and data analysis. It also enjoys equal patronage among the novices and the Experts since it can be used and accessed easily.



Welcome to Colab!

Explore the Gemini API

The Gemini API gives you access to Gemini models created by Google DeepMind. Gemini models are built from the ground up to be multimodal, so you can reason seamlessly across text, images, code, and audio.

How to get started

1. Go to Google AI Studio and log in with your Google account.
2. Create an API key.
3. Use a quickstart for Python, or call the REST API using curl.

Explore use cases

- Create a marketing campaign
- Analyze audio recordings
- Use System instructions in chat

To learn more, check out the Gemini cookbook or visit the Gemini API documentation.

Figure 3.2: Google Colab

4 Libraries Overview

cv2 (OpenCV): An image and video library used in computer visions which are needed in image and video operations. Some of the functions from cv2 that are used to sharpen images, detect objects, and several other conversions.
Math: Has basic mathematical functions like trigonometric, logarithmic, constant functions and many others. It is employed in the actual arithmetic manipulation in the scripts that you are developing for your software.
Time: It allows for time-related operations that may include the setting as to how much time a particular code will take or even incorporating time in your code.
random: Used in activities such as; production of random numbers or any random work that you may think of. This type is usually used when the model is to be tested by feeding it with many sets of data or when some arbitrary data is required.
numpy (np): Of Python, the numerical operations library that can be considered as an essential element of it. Several functions for array manipulation are provided including matrices, and an almost limitless host of mathematical preliminary operations to optimize control of these structures.
pandas (pd): It can be defined as a highly effective data post-processing and analysis instrument. Some of them include Data Frames that assist in working and operating on formatted data easily.
copy: It provides the services to create new objects with or without the operations on the objects' contents. Used when it is required to lock mutable objects which should not be altered.
seaborn (sns): An aggressive statistical data visualization library that is based upon matplotlib. They are used for getting a lower level of interface for drawing good looking and informative statistical graphics.
PIL (Python Imaging Library): Originally developed as Pillow, it is used for handling various kinds of images and the files handling and storing.
matplotlib. pyplot (plt): A plotting library embedded in the vehicle of creating static, interactive and animated graphics in Python. This provides MATLAB like feel to create and generate plots and charts.
google. colab. patches: Comprises of a function called cv2_imshow which relates exclusively to Google colab and will display Open CV image within the colab notebooks. (colab.research.google.com, n.d.)

5 Usage

1. Set Up Notebooks: Click on the notebooks in order to open them in Jupyter Notebook. It is better to perform simple variations of one type before switching to a combination of another type, such as tinker-henon followed by tinker-duffing and duffing-henon.
2. Run the Notebooks: Do all the cells of one notebook one by one. That all cells must be executed to keep all bounded input data in order to continue the computations of the program.
3. Encrypt an Image:
- Load an image using the described methods.
- These are the steps to encrypt the image using the chaotic mappings:

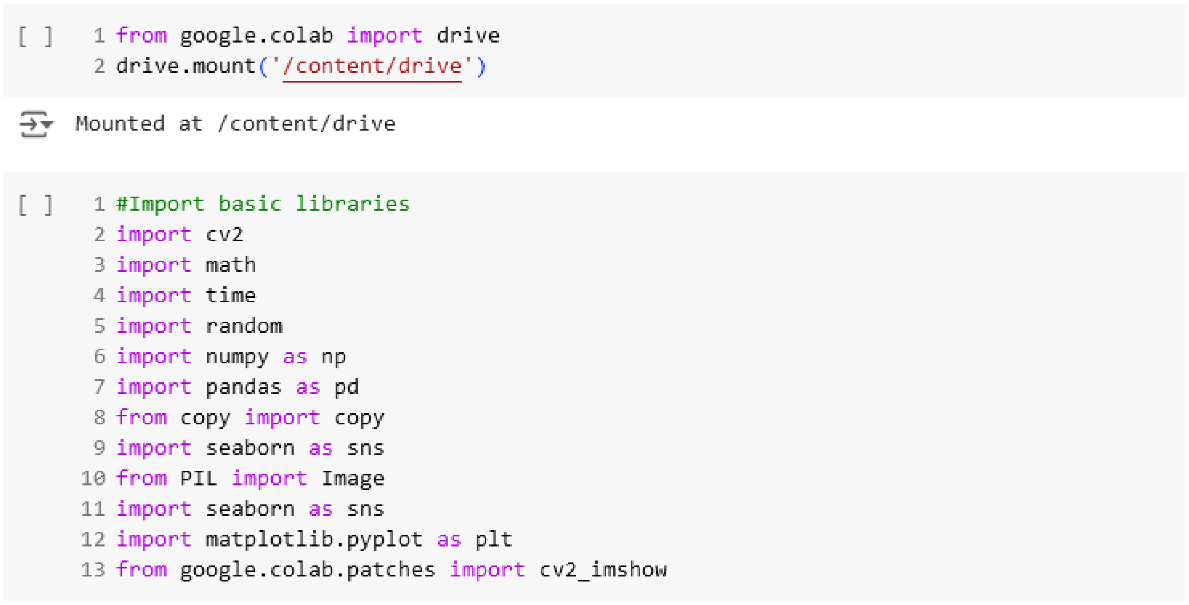
4. Save the Encrypted Image:

- The result to be produced will be advanced in the defined folder. See to it that this path is correctly set in the duffing-henon notebook.

6 Project Implementation

6.1 Henon and Duffing

6.1.1 Importing Libraries

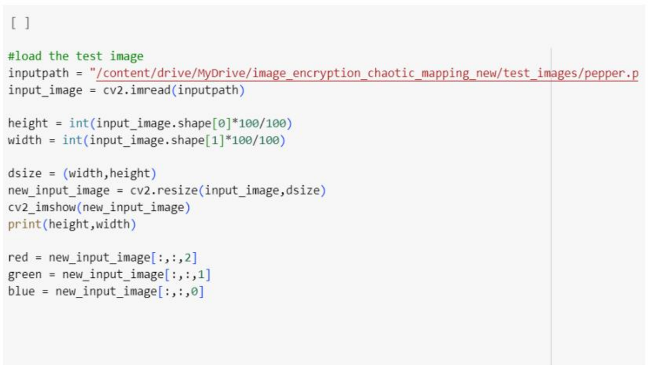


```
[ ]
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
[ ]
#Import basic libraries
import cv2
import math
import time
import random
import numpy as np
import pandas as pd
from copy import copy
import seaborn as sns
from PIL import Image
import seaborn as sns
import matplotlib.pyplot as plt
from google.colab.patches import cv2_imshow
```

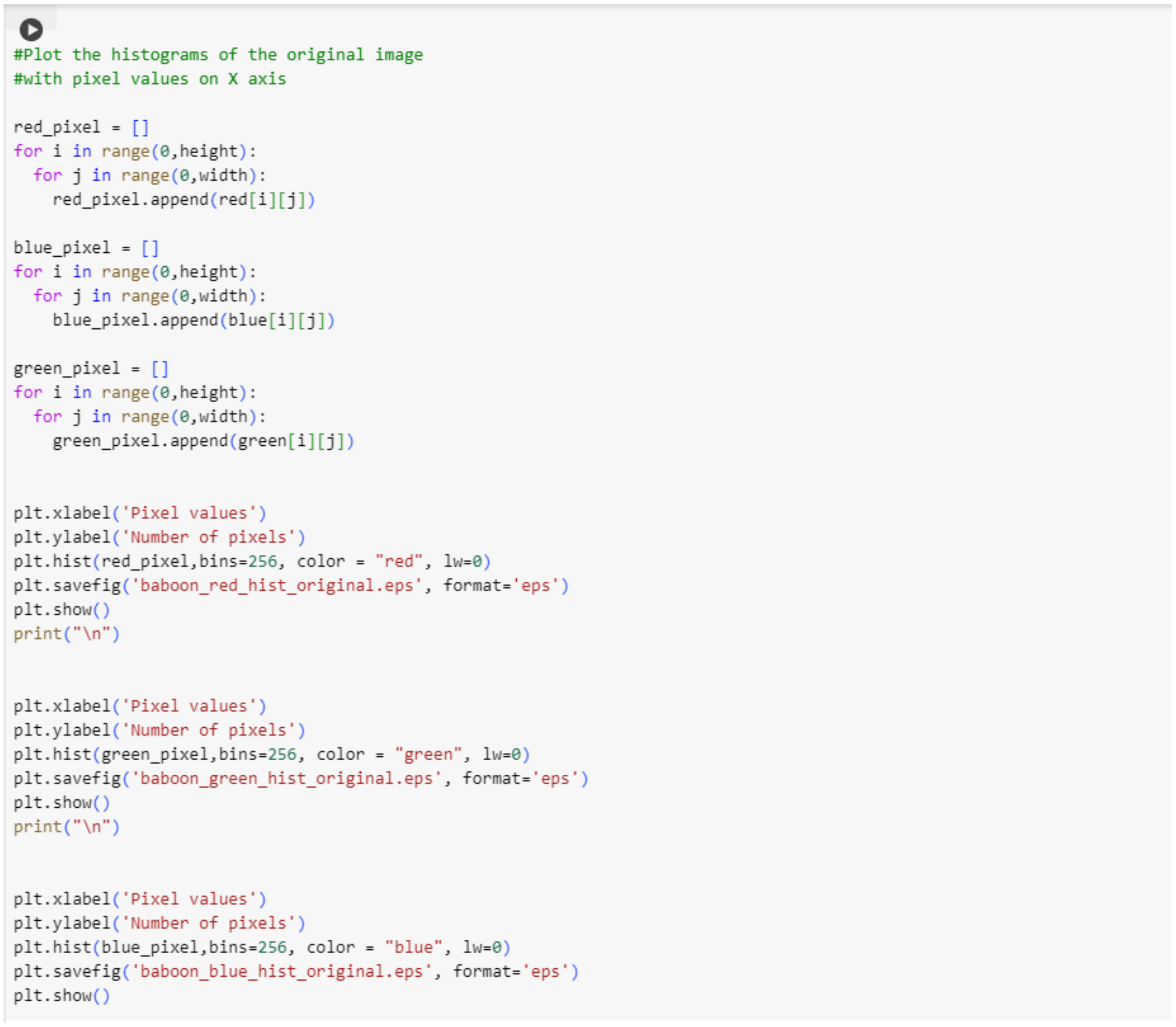
6.1.2 Image Loading

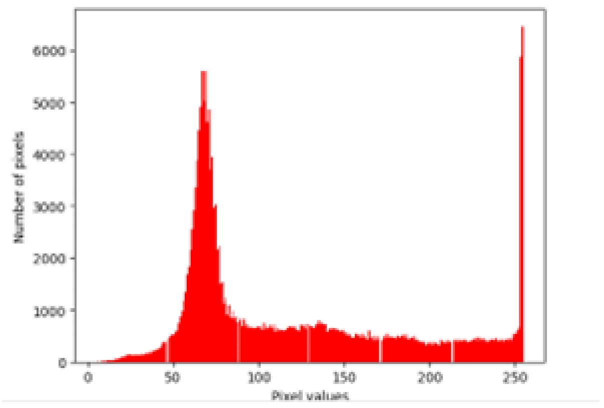


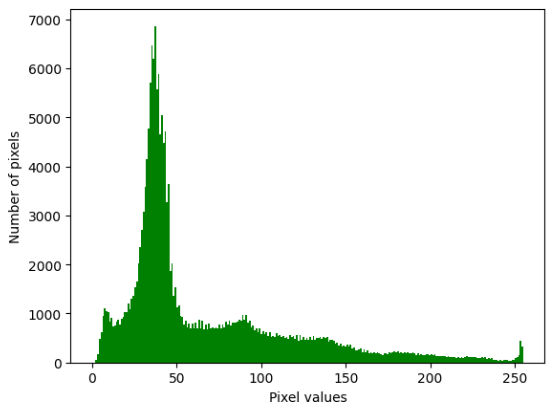
```
[ ]

#load the test image
inputpath = "/content/drive/MyDrive/image_encryption_chaotic_mapping_new/test_images/pepper.p
input_image = cv2.imread(inputpath)

height = int(input_image.shape[0]*100/100)
width = int(input_image.shape[1]*100/100)

dsize = (width,height)
new_input_image = cv2.resize(input_image,dsize)
cv2_imshow(new_input_image)
print(height,width)

red = new_input_image[:,:,2]
green = new_input_image[:,:,1]
blue = new_input_image[:,:,0]
```

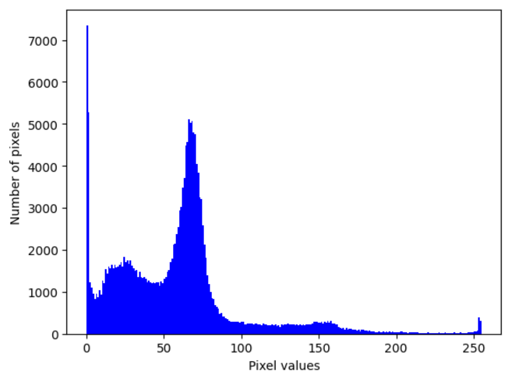
```
384 512
```

6.2 Histogram of original image


```
#Plot the histograms of the original image
#with pixel values on X axis

red_pixel = []
for i in range(0,height):
  for j in range(0,width):
    red_pixel.append(red[i][j])

blue_pixel = []
for i in range(0,height):
  for j in range(0,width):
    blue_pixel.append(blue[i][j])

green_pixel = []
for i in range(0,height):
  for j in range(0,width):
    green_pixel.append(green[i][j])


plt.xlabel('Pixel values')
plt.ylabel('Number of pixels')
plt.hist(red_pixel,bins=256, color = "red", lw=0)
plt.savefig('baboon_red_hist_original.eps', format='eps')
plt.show()
print("\n")


plt.xlabel('Pixel values')
plt.ylabel('Number of pixels')
plt.hist(green_pixel,bins=256, color = "green", lw=0)
plt.savefig('baboon_green_hist_original.eps', format='eps')
plt.show()
print("\n")


plt.xlabel('Pixel values')
plt.ylabel('Number of pixels')
plt.hist(blue_pixel,bins=256, color = "blue", lw=0)
plt.savefig('baboon_blue_hist_original.eps', format='eps')
plt.show()
```

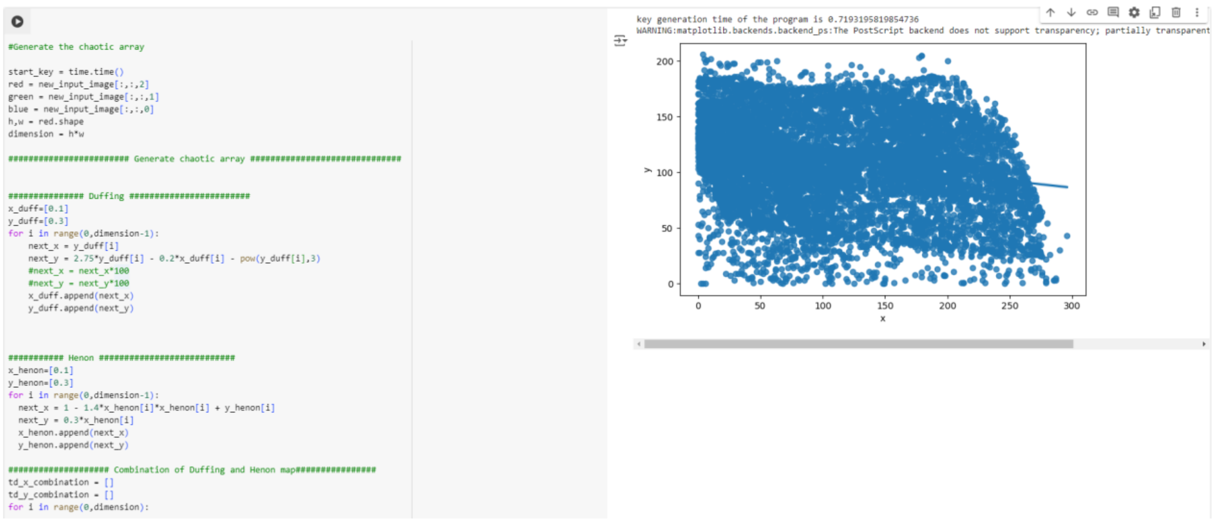






6.3 Generating the chaotic array -

Generating the chaotic array with output.

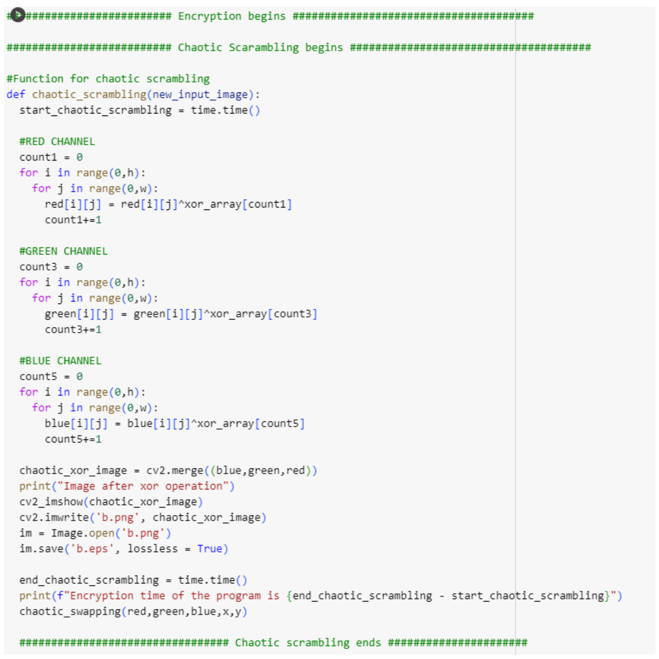


```
#Generate the chaotic array

start_key = time.time()
red = new_input_image[:,:,2]
green = new_input_image[:,:,1]
blue = new_input_image[:,:,0]
h,w = red.shape
dimension = h*w

######################## Generate chaotic array ##############################


############### Duffing ########################
x_duff=[0.1]
y_duff=[0.3]
for i in range(0,dimension-1):
    next_x = y_duff[i]
    next_y = 2.75*y_duff[i] - 0.2*x_duff[i] - pow(y_duff[i],3)
    #next_x = next_x*100
    #next_y = next_y*100
    x_duff.append(next_x)
    y_duff.append(next_y)



########### Henon ###########################
x_henon=[0.1]
y_henon=[0.3]
for i in range(0,dimension-1):
  next_x = 1 - 1.4*x_henon[i]*x_henon[i] + y_henon[i]
  next_y = 0.3*x_henon[i]
  x_henon.append(next_x)
  y_henon.append(next_y)

#################### Combination of Duffing and Henon map################
td_x_combination = []
td_y_combination = []
for i in range(0,dimension):
```

```
key generation time of the program is 0.7193195819854736
WARNING:matplotlib.backends.backend_ps:The PostScript backend does not support transparency; partially transparent
```

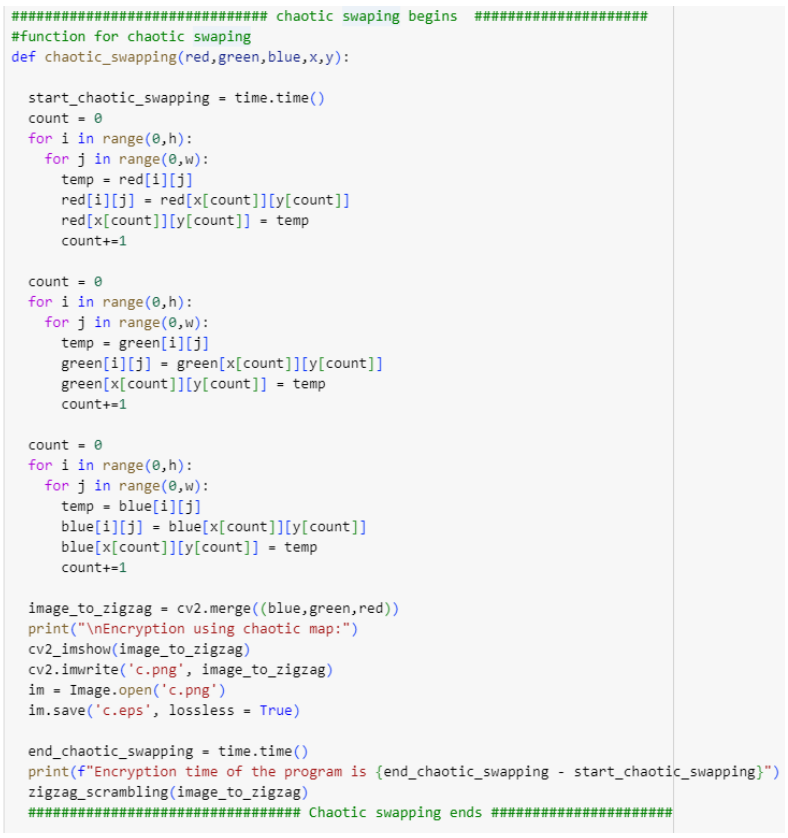
6.4 Encryption Begins –

Chaotic Scrambling



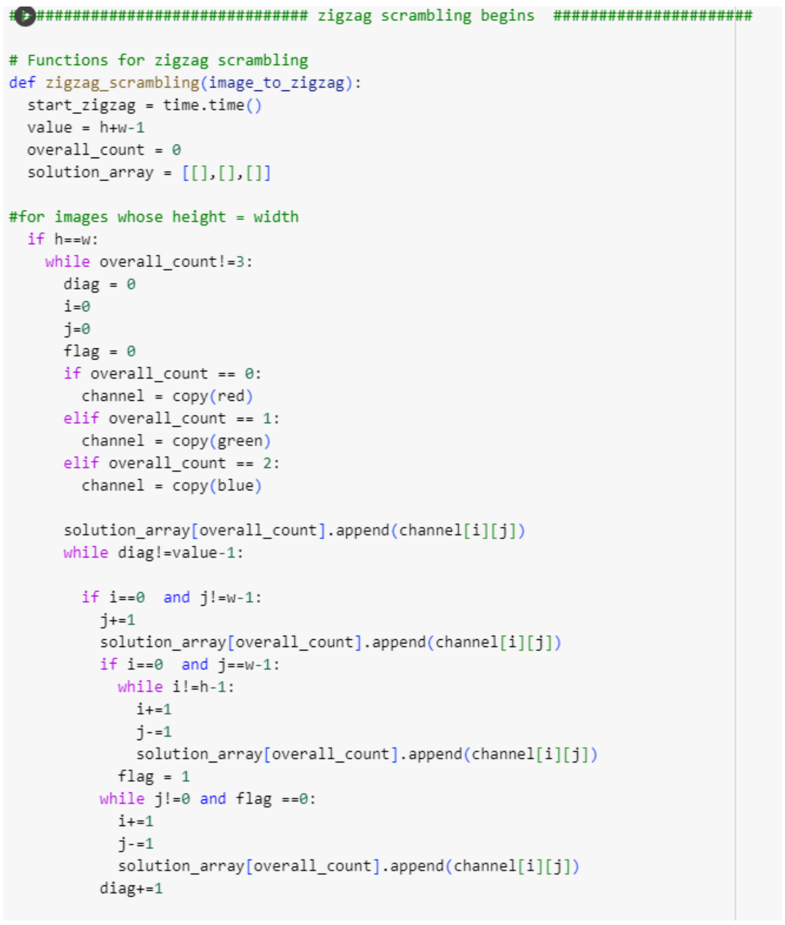
```
######################### Encryption begins ######################################

######################### Chaotic Scarambling begins ######################################

#Function for chaotic scrambling
def chaotic_scrambling(new_input_image):
  start_chaotic_scrambling = time.time()

  #RED CHANNEL
  count1 = 0
  for i in range(0,h):
    for j in range(0,w):
      red[i][j] = red[i][j]^xor_array[count1]
      count1+=1

  #GREEN CHANNEL
  count3 = 0
  for i in range(0,h):
    for j in range(0,w):
      green[i][j] = green[i][j]^xor_array[count3]
      count3+=1

  #BLUE CHANNEL
  count5 = 0
  for i in range(0,h):
    for j in range(0,w):
      blue[i][j] = blue[i][j]^xor_array[count5]
      count5+=1

  chaotic_xor_image = cv2.merge((blue,green,red))
  print("Image after xor operation")
  cv2_imshow(chaotic_xor_image)
  cv2.imwrite('b.png', chaotic_xor_image)
  im = Image.open('b.png')
  im.save('b.eps', lossless = True)

  end_chaotic_scrambling = time.time()
  print(f"Encryption time of the program is {end_chaotic_scrambling - start_chaotic_scrambling}")
  chaotic_swapping(red,green,blue,x,y)

  ################################# Chaotic scrambling ends #####################
```

chaotic swapping –



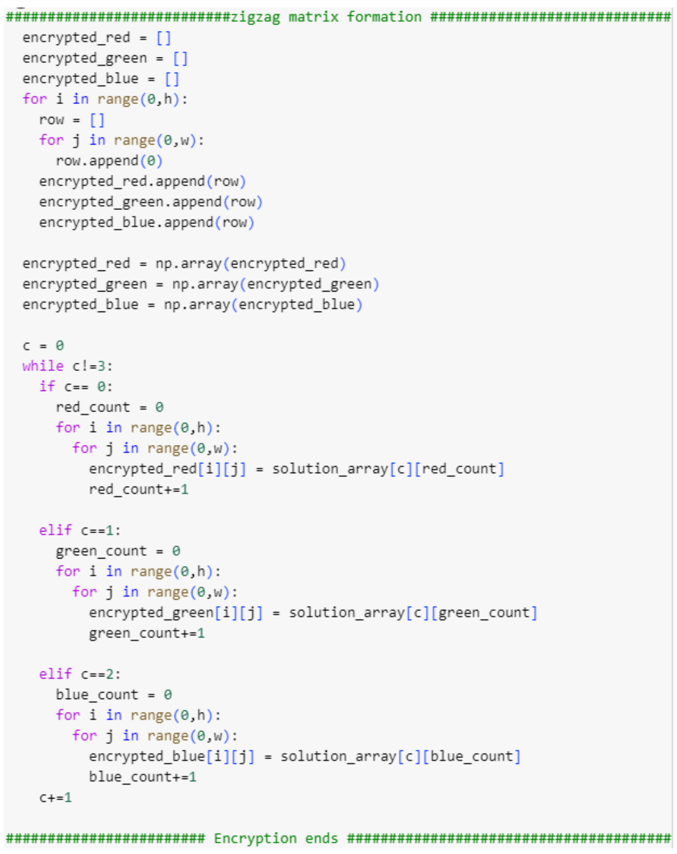
```
############################### chaotic swaping begins  #####################
#function for chaotic swaping
def chaotic_swapping(red,green,blue,x,y):

  start_chaotic_swapping = time.time()
  count = 0
  for i in range(0,h):
    for j in range(0,w):
      temp = red[i][j]
      red[i][j] = red[x[count]][y[count]]
      red[x[count]][y[count]] = temp
      count+=1

  count = 0
  for i in range(0,h):
    for j in range(0,w):
      temp = green[i][j]
      green[i][j] = green[x[count]][y[count]]
      green[x[count]][y[count]] = temp
      count+=1

  count = 0
  for i in range(0,h):
    for j in range(0,w):
      temp = blue[i][j]
      blue[i][j] = blue[x[count]][y[count]]
      blue[x[count]][y[count]] = temp
      count+=1

  image_to_zigzag = cv2.merge((blue,green,red))
  print("\nEncryption using chaotic map:")
  cv2_imshow(image_to_zigzag)
  cv2.imwrite('c.png', image_to_zigzag)
  im = Image.open('c.png')
  im.save('c.eps', lossless = True)

  end_chaotic_swapping = time.time()
  print(f"Encryption time of the program is {end_chaotic_swapping - start_chaotic_swapping}")
  zigzag_scrambling(image_to_zigzag)
  ################################# Chaotic swapping ends #####################
```

Zigzag Scrambling -



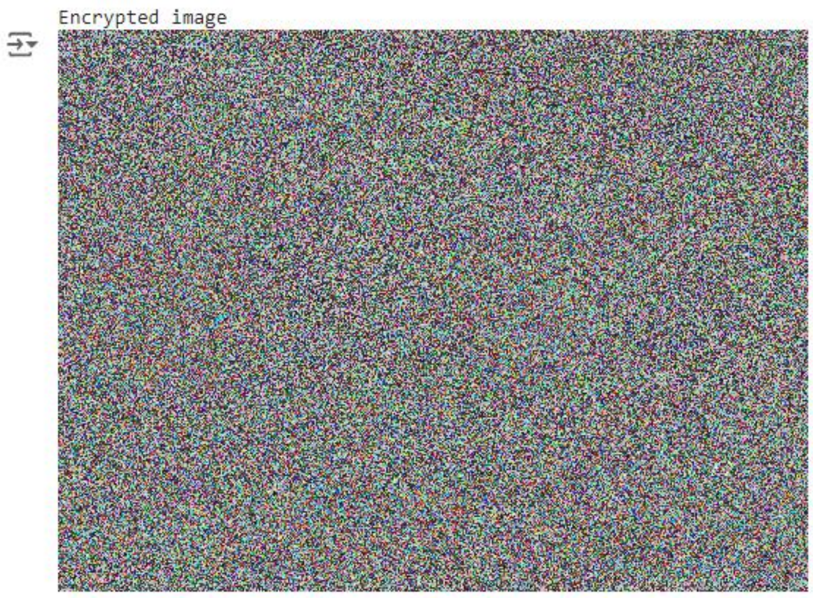
```
################################# zigzag scrambling begins  ######################

# Functions for zigzag scrambling
def zigzag_scrambling(image_to_zigzag):
  start_zigzag = time.time()
  value = h+w-1
  overall_count = 0
  solution_array = [[],[],[]]

#for images whose height = width
  if h==w:
    while overall_count!=3:
      diag = 0
      i=0
      j=0
      flag = 0
      if overall_count == 0:
        channel = copy(red)
      elif overall_count == 1:
        channel = copy(green)
      elif overall_count == 2:
        channel = copy(blue)

      solution_array[overall_count].append(channel[i][j])
      while diag!=value-1:

        if i==0  and j!=w-1:
          j+=1
          solution_array[overall_count].append(channel[i][j])
          if i==0  and j==w-1:
            while i!=h-1:
              i+=1
              j-=1
              solution_array[overall_count].append(channel[i][j])
            flag = 1
          while j!=0 and flag ==0:
            i+=1
            j-=1
            solution_array[overall_count].append(channel[i][j])
          diag+=1
```

Then, Zigzag Matrix Formation-



```
##########################zigzag matrix formation ############################
  encrypted_red = []
  encrypted_green = []
  encrypted_blue = []
  for i in range(0,h):
    row = []
    for j in range(0,w):
      row.append(0)
    encrypted_red.append(row)
    encrypted_green.append(row)
    encrypted_blue.append(row)

  encrypted_red = np.array(encrypted_red)
  encrypted_green = np.array(encrypted_green)
  encrypted_blue = np.array(encrypted_blue)

  c = 0
  while c!=3:
    if c== 0:
      red_count = 0
      for i in range(0,h):
        for j in range(0,w):
          encrypted_red[i][j] = solution_array[c][red_count]
          red_count+=1

    elif c==1:
      green_count = 0
      for i in range(0,h):
        for j in range(0,w):
          encrypted_green[i][j] = solution_array[c][green_count]
          green_count+=1

    elif c==2:
      blue_count = 0
      for i in range(0,h):
        for j in range(0,w):
          encrypted_blue[i][j] = solution_array[c][blue_count]
          blue_count+=1
    c+=1

####################### Encryption ends ######################################
```

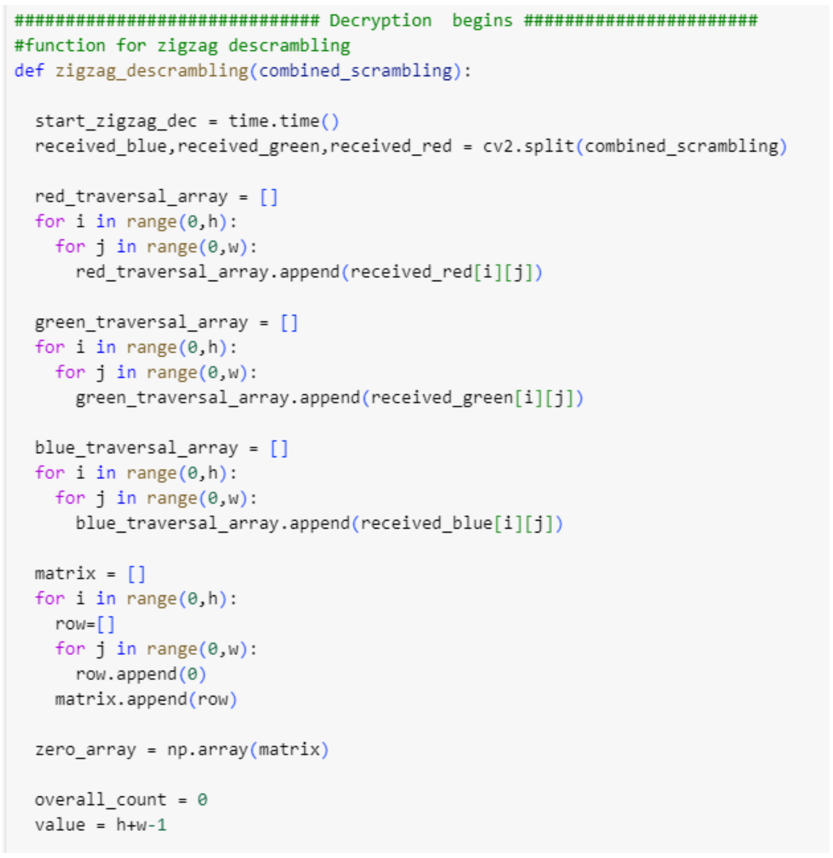
Here Get the Encrypted Image –



Now, Decryption process begins –

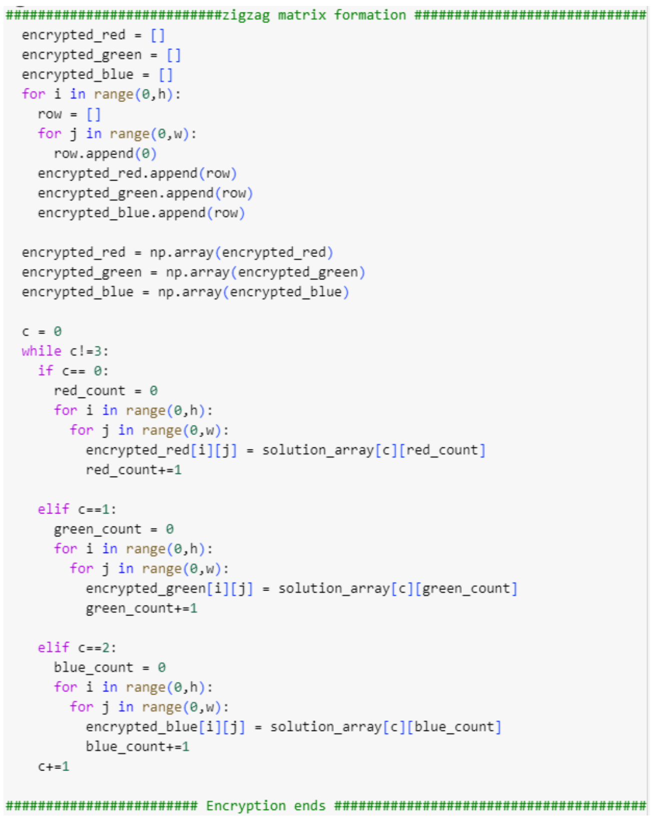
Here start reverse process like start from zigzag decryption to original image

A) Zigzag descrambling –



```
############################### Decryption  begins #######################
#function for zigzag descrambling
def zigzag_descrambling(combined_scrambling):

  start_zigzag_dec = time.time()
  received_blue,received_green,received_red = cv2.split(combined_scrambling)

  red_traversal_array = []
  for i in range(0,h):
    for j in range(0,w):
      red_traversal_array.append(received_red[i][j])

  green_traversal_array = []
  for i in range(0,h):
    for j in range(0,w):
      green_traversal_array.append(received_green[i][j])

  blue_traversal_array = []
  for i in range(0,h):
    for j in range(0,w):
      blue_traversal_array.append(received_blue[i][j])

  matrix = []
  for i in range(0,h):
    row=[]
    for j in range(0,w):
      row.append(0)
    matrix.append(row)

  zero_array = np.array(matrix)

  overall_count = 0
  value = h+w-1
```

Then, Zigzag Matrix Formation-



```
##########################zigzag matrix formation ############################
  encrypted_red = []
  encrypted_green = []
  encrypted_blue = []
  for i in range(0,h):
    row = []
    for j in range(0,w):
      row.append(0)
    encrypted_red.append(row)
    encrypted_green.append(row)
    encrypted_blue.append(row)

  encrypted_red = np.array(encrypted_red)
  encrypted_green = np.array(encrypted_green)
  encrypted_blue = np.array(encrypted_blue)

  c = 0
  while c!=3:
    if c== 0:
      red_count = 0
      for i in range(0,h):
        for j in range(0,w):
          encrypted_red[i][j] = solution_array[c][red_count]
          red_count+=1

    elif c==1:
      green_count = 0
      for i in range(0,h):
        for j in range(0,w):
          encrypted_green[i][j] = solution_array[c][green_count]
          green_count+=1

    elif c==2:
      blue_count = 0
      for i in range(0,h):
        for j in range(0,w):
          encrypted_blue[i][j] = solution_array[c][blue_count]
          blue_count+=1
    c+=1

####################### Encryption ends ######################################
```

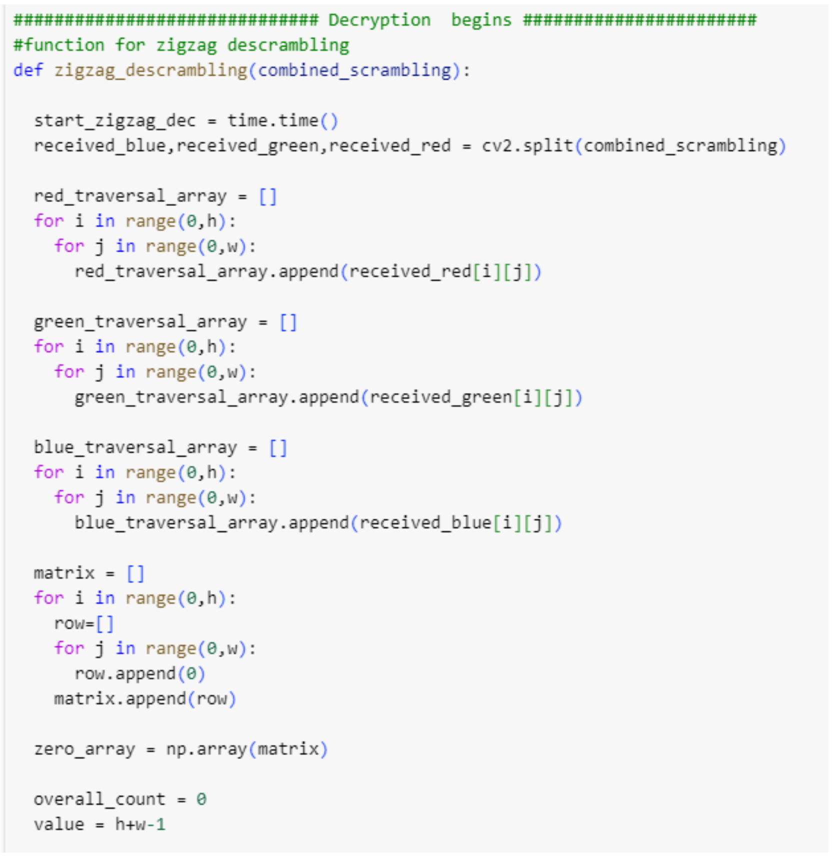
Encrypted Image -

Encrypted image

Now, Decryption process begins –

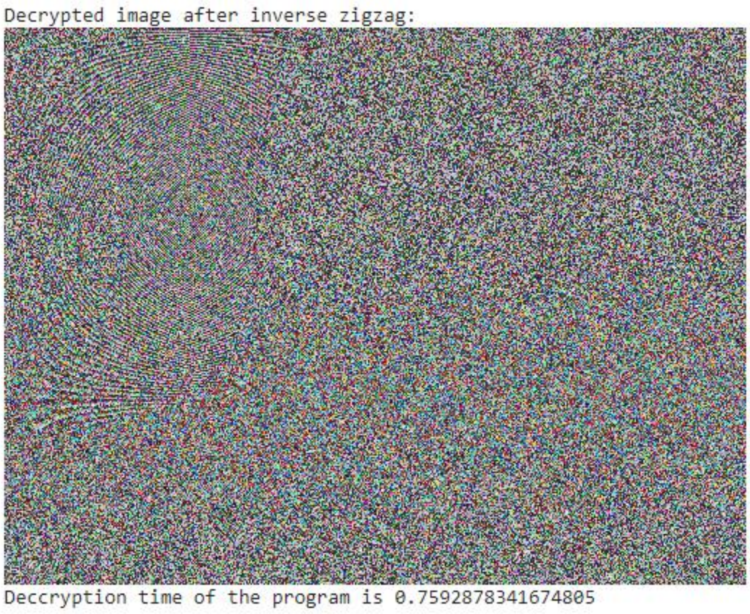
Here start reverse process like start from zigzag decryption to original image

A) Zigzag descrambling -



```
############################## Decryption  begins #######################
#function for zigzag descrambling
def zigzag_descrambling(combined_scrambling):

  start_zigzag_dec = time.time()
  received_blue,received_green,received_red = cv2.split(combined_scrambling)

  red_traversal_array = []
  for i in range(0,h):
    for j in range(0,w):
      red_traversal_array.append(received_red[i][j])

  green_traversal_array = []
  for i in range(0,h):
    for j in range(0,w):
      green_traversal_array.append(received_green[i][j])

  blue_traversal_array = []
  for i in range(0,h):
    for j in range(0,w):
      blue_traversal_array.append(received_blue[i][j])

  matrix = []
  for i in range(0,h):
    row=[]
    for j in range(0,w):
      row.append(0)
    matrix.append(row)

  zero_array = np.array(matrix)

  overall_count = 0
  value = h+w-1
```

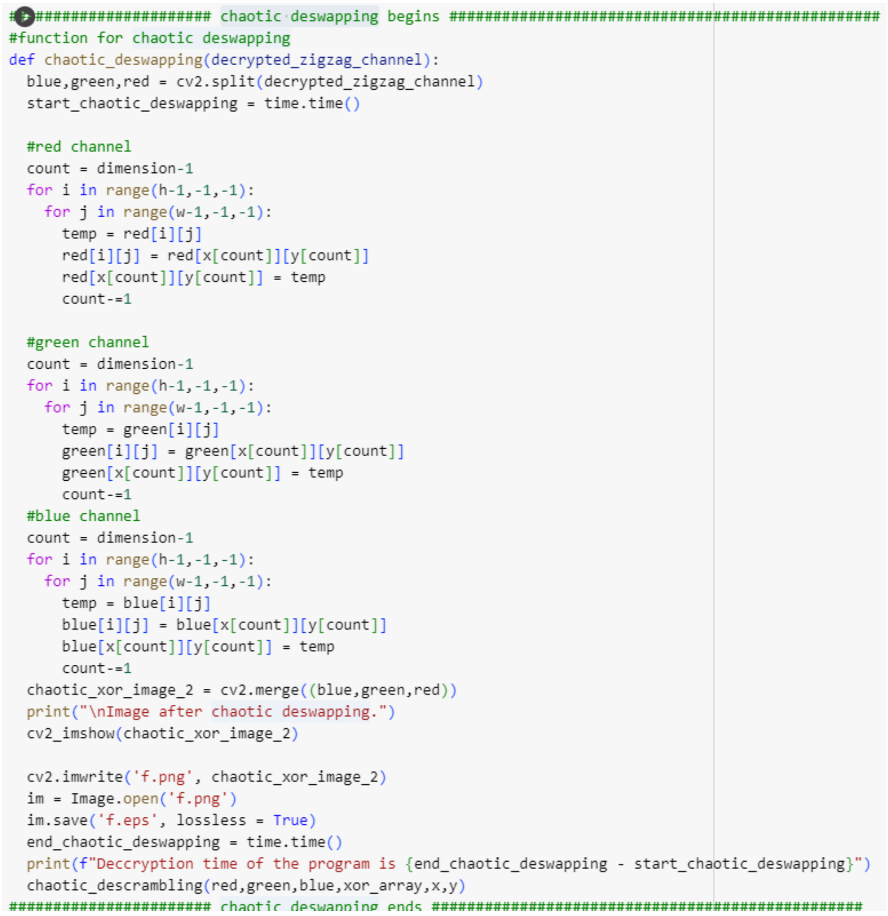
After zigzag descrambling image –



```
Decrypted image after inverse zigzag:
```

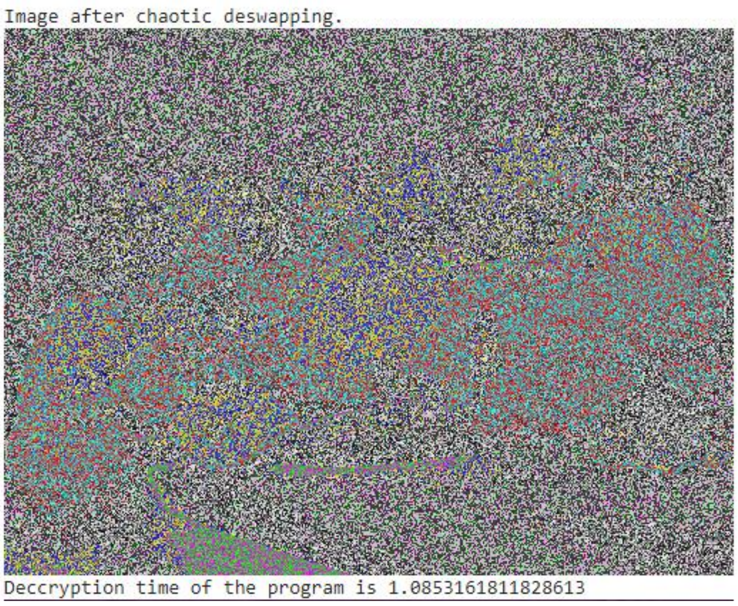
```
Deccryption time of the program is 0.7592878341674805
```

B) Chaotic deswapping –



```
##################### chaotic deswapping begins ##################################################
#function for chaotic deswapping
def chaotic_deswapping(decrypted_zigzag_channel):
  blue,green,red = cv2.split(decrypted_zigzag_channel)
  start_chaotic_deswapping = time.time()

  #red channel
  count = dimension-1
  for i in range(h-1,-1,-1):
    for j in range(w-1,-1,-1):
      temp = red[i][j]
      red[i][j] = red[x[count]][y[count]]
      red[x[count]][y[count]] = temp
      count-=1

  #green channel
  count = dimension-1
  for i in range(h-1,-1,-1):
    for j in range(w-1,-1,-1):
      temp = green[i][j]
      green[i][j] = green[x[count]][y[count]]
      green[x[count]][y[count]] = temp
      count-=1
  #blue channel
  count = dimension-1
  for i in range(h-1,-1,-1):
    for j in range(w-1,-1,-1):
      temp = blue[i][j]
      blue[i][j] = blue[x[count]][y[count]]
      blue[x[count]][y[count]] = temp
      count-=1
  chaotic_xor_image_2 = cv2.merge((blue,green,red))
  print("\nImage after chaotic deswapping.")
  cv2_imshow(chaotic_xor_image_2)

  cv2.imwrite('f.png', chaotic_xor_image_2)
  im = Image.open('f.png')
  im.save('f.eps', lossless = True)
  end_chaotic_deswapping = time.time()
  print(f"Deccryption time of the program is {end_chaotic_deswapping - start_chaotic_deswapping}")
  chaotic_descrambling(red,green,blue,xor_array,x,y)
####################### chaotic deswapping ends ###################################################
```

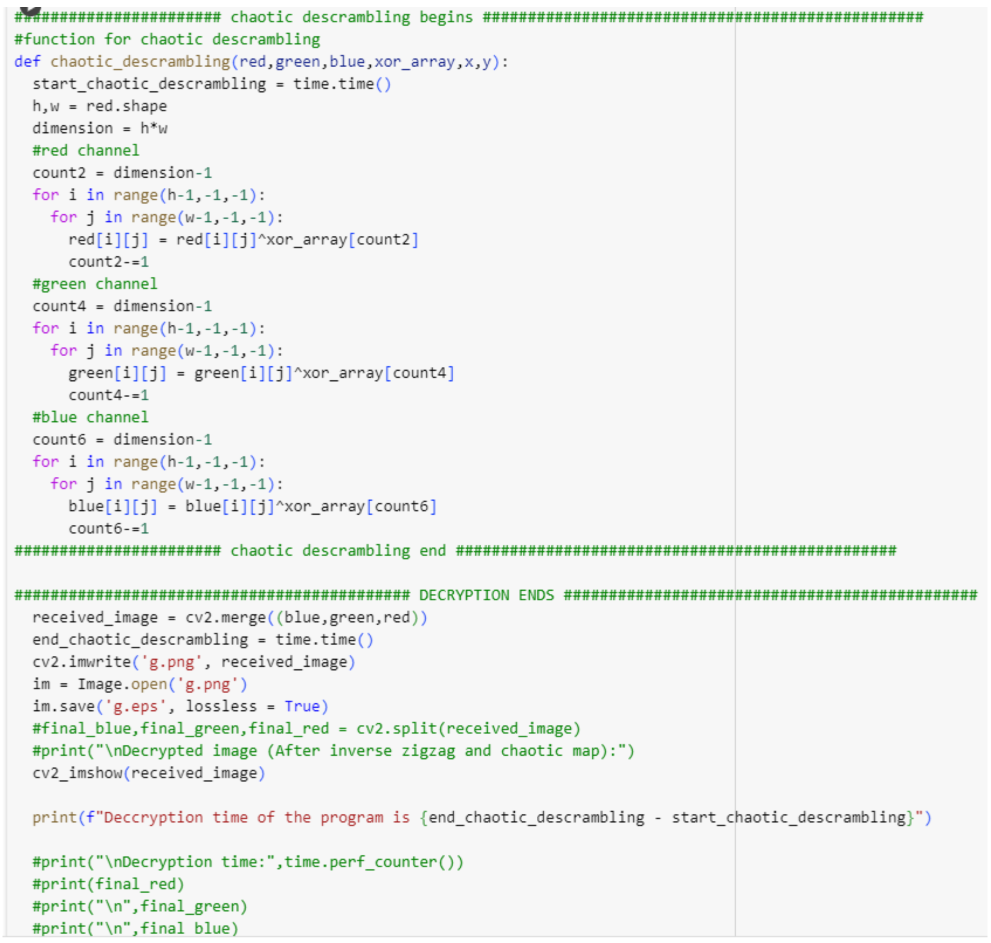
After deswapping image –



```
Image after chaotic deswapping.
```

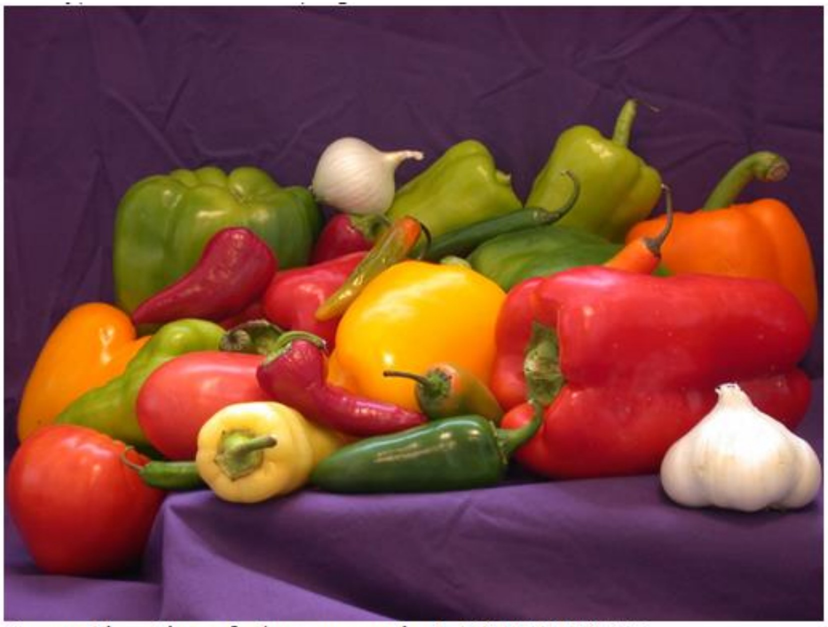
```
Deccryption time of the program is 1.0853161811828613
```

C) Chaotic Descrambling –



```
####################### chaotic descrambling begins ##################################################
#function for chaotic descrambling
def chaotic_descrambling(red,green,blue,xor_array,x,y):
  start_chaotic_descrambling = time.time()
  h,w = red.shape
  dimension = h*w
  #red channel
  count2 = dimension-1
  for i in range(h-1,-1,-1):
    for j in range(w-1,-1,-1):
      red[i][j] = red[i][j]^xor_array[count2]
      count2-=1
  #green channel
  count4 = dimension-1
  for i in range(h-1,-1,-1):
    for j in range(w-1,-1,-1):
      green[i][j] = green[i][j]^xor_array[count4]
      count4-=1
  #blue channel
  count6 = dimension-1
  for i in range(h-1,-1,-1):
    for j in range(w-1,-1,-1):
      blue[i][j] = blue[i][j]^xor_array[count6]
      count6-=1
####################### chaotic descrambling end ##################################################

############################################ DECRYPTION ENDS ##############################################
  received_image = cv2.merge((blue,green,red))
  end_chaotic_descrambling = time.time()
  cv2.imwrite('g.png', received_image)
  im = Image.open('g.png')
  im.save('g.eps', lossless = True)
  #final_blue,final_green,final_red = cv2.split(received_image)
  #print("\nDecrypted image (After inverse zigzag and chaotic map):")
  cv2_imshow(received_image)

  print(f"Deccryption time of the program is {end_chaotic_descrambling - start_chaotic_descrambling}")

  #print("\nDecryption time:",time.perf_counter())
  #print(final_red)
  #print("\n",final_green)
  #print("\n",final blue)
```

After Chaotic Descrambling Image



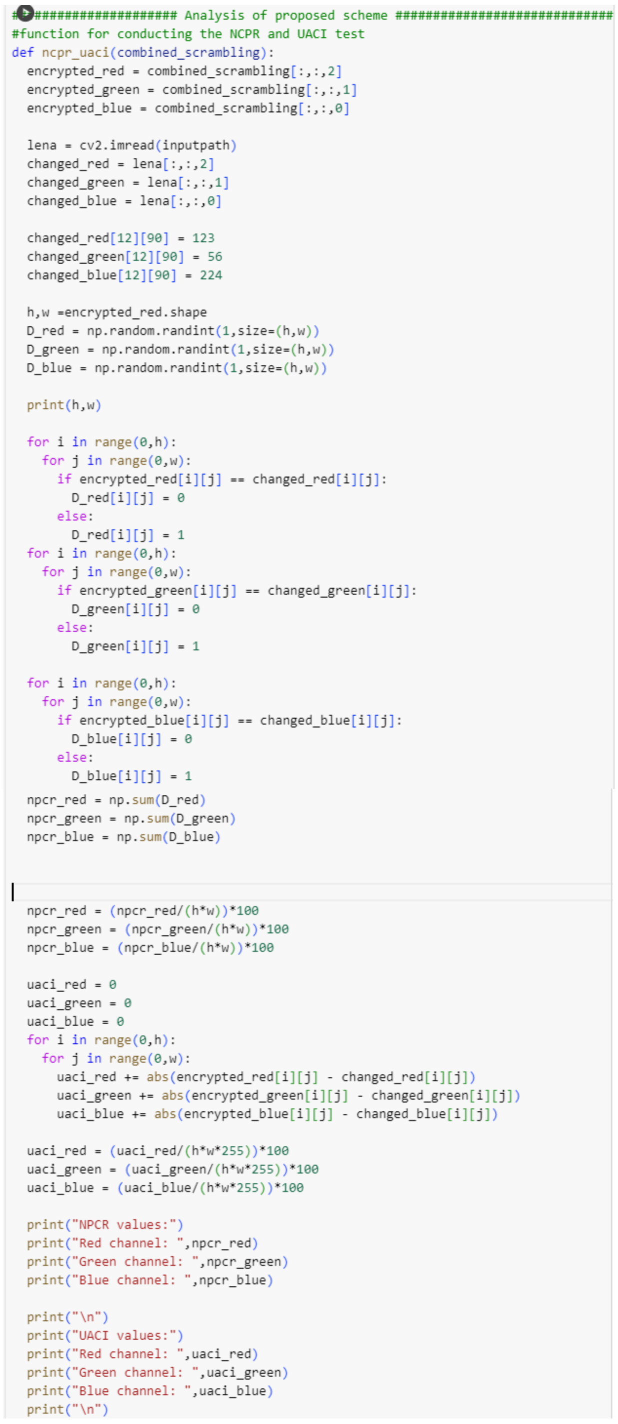
```
Deccryption time of the program is 1.718552827835083
```

Here get the original image.

Analysis of Proposed Scheme –

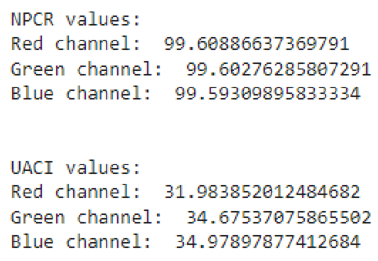
Here conducting the various tests to compare with original image.

A) NPCR and UACI test-



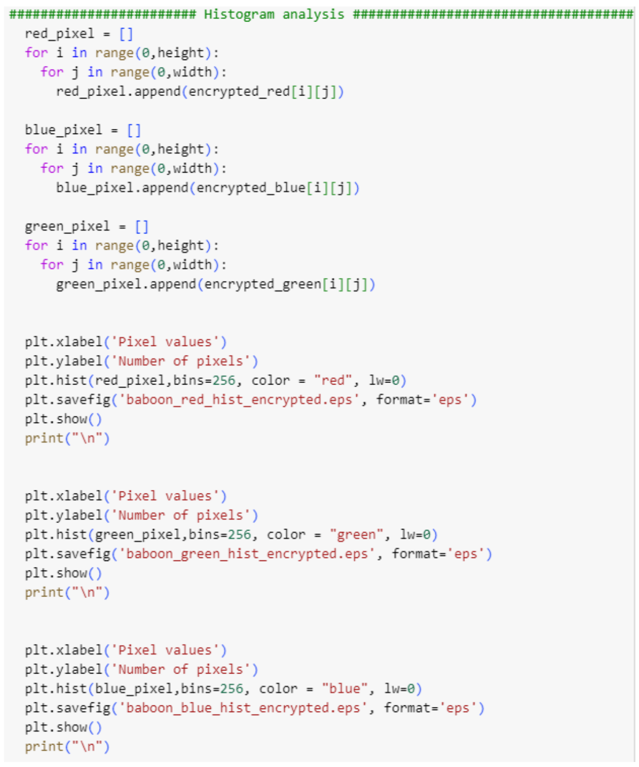
```
#################### Analysis of proposed scheme ############################
#function for conducting the NCPR and UACI test
def ncpr_uaci(combined_scrambling):
  encrypted_red = combined_scrambling[:,:,2]
  encrypted_green = combined_scrambling[:,:,1]
  encrypted_blue = combined_scrambling[:,:,0]

  lena = cv2.imread(inputpath)
  changed_red = lena[:,:,2]
  changed_green = lena[:,:,1]
  changed_blue = lena[:,:,0]

  changed_red[12][90] = 123
  changed_green[12][90] = 56
  changed_blue[12][90] = 224

  h,w =encrypted_red.shape
  D_red = np.random.randint(1,size=(h,w))
  D_green = np.random.randint(1,size=(h,w))
  D_blue = np.random.randint(1,size=(h,w))

  print(h,w)

  for i in range(0,h):
    for j in range(0,w):
      if encrypted_red[i][j] == changed_red[i][j]:
        D_red[i][j] = 0
      else:
        D_red[i][j] = 1
  for i in range(0,h):
    for j in range(0,w):
      if encrypted_green[i][j] == changed_green[i][j]:
        D_green[i][j] = 0
      else:
        D_green[i][j] = 1

  for i in range(0,h):
    for j in range(0,w):
      if encrypted_blue[i][j] == changed_blue[i][j]:
        D_blue[i][j] = 0
      else:
        D_blue[i][j] = 1
  npcr_red = np.sum(D_red)
  npcr_green = np.sum(D_green)
  npcr_blue = np.sum(D_blue)


  npcr_red = (npcr_red/(h*w))*100
  npcr_green = (npcr_green/(h*w))*100
  npcr_blue = (npcr_blue/(h*w))*100

  uaci_red = 0
  uaci_green = 0
  uaci_blue = 0
  for i in range(0,h):
    for j in range(0,w):
      uaci_red += abs(encrypted_red[i][j] - changed_red[i][j])
      uaci_green += abs(encrypted_green[i][j] - changed_green[i][j])
      uaci_blue += abs(encrypted_blue[i][j] - changed_blue[i][j])

  uaci_red = (uaci_red/(h*w*255))*100
  uaci_green = (uaci_green/(h*w*255))*100
  uaci_blue = (uaci_blue/(h*w*255))*100

  print("NPCR values:")
  print("Red channel: ",npcr_red)
  print("Green channel: ",npcr_green)
  print("Blue channel: ",npcr_blue)

  print("\n")
  print("UACI values:")
  print("Red channel: ",uaci_red)
  print("Green channel: ",uaci_green)
  print("Blue channel: ",uaci_blue)
  print("\n")
```

Result –



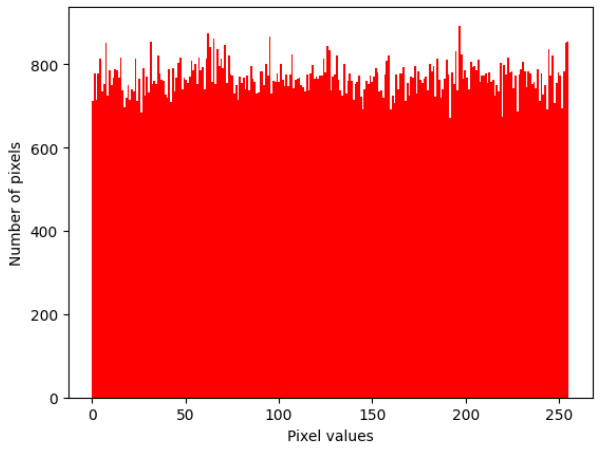
```
NPCR values:
Red channel:  99.60886637369791
Green channel:  99.60276285807291
Blue channel:  99.59309895833334


UACI values:
Red channel:  31.983852012484682
Green channel:  34.67537075865502
Blue channel:  34.97897877412684
```

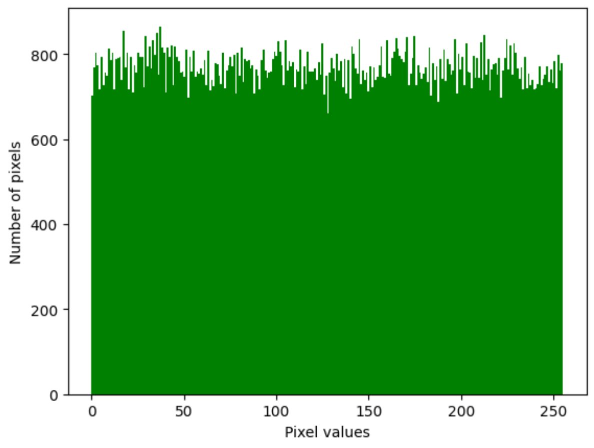
B) Histogram analysis –

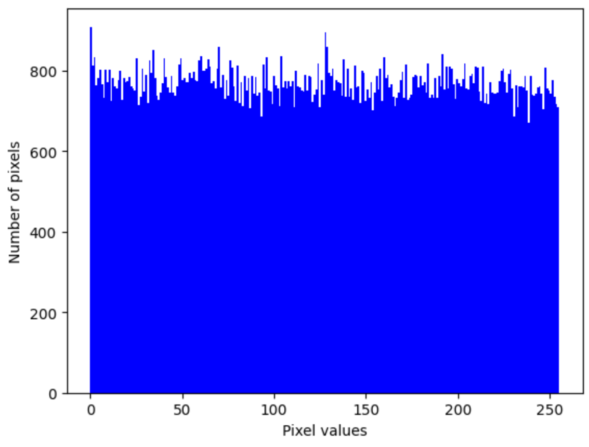


```
######################## Histogram analysis ####################################
  red_pixel = []
  for i in range(0,height):
    for j in range(0,width):
      red_pixel.append(encrypted_red[i][j])

  blue_pixel = []
  for i in range(0,height):
    for j in range(0,width):
      blue_pixel.append(encrypted_blue[i][j])

  green_pixel = []
  for i in range(0,height):
    for j in range(0,width):
      green_pixel.append(encrypted_green[i][j])


  plt.xlabel('Pixel values')
  plt.ylabel('Number of pixels')
  plt.hist(red_pixel,bins=256, color = "red", lw=0)
  plt.savefig('baboon_red_hist_encrypted.eps', format='eps')
  plt.show()
  print("\n")


  plt.xlabel('Pixel values')
  plt.ylabel('Number of pixels')
  plt.hist(green_pixel,bins=256, color = "green", lw=0)
  plt.savefig('baboon_green_hist_encrypted.eps', format='eps')
  plt.show()
  print("\n")


  plt.xlabel('Pixel values')
  plt.ylabel('Number of pixels')
  plt.hist(blue_pixel,bins=256, color = "blue", lw=0)
  plt.savefig('baboon_blue_hist_encrypted.eps', format='eps')
  plt.show()
  print("\n")
```

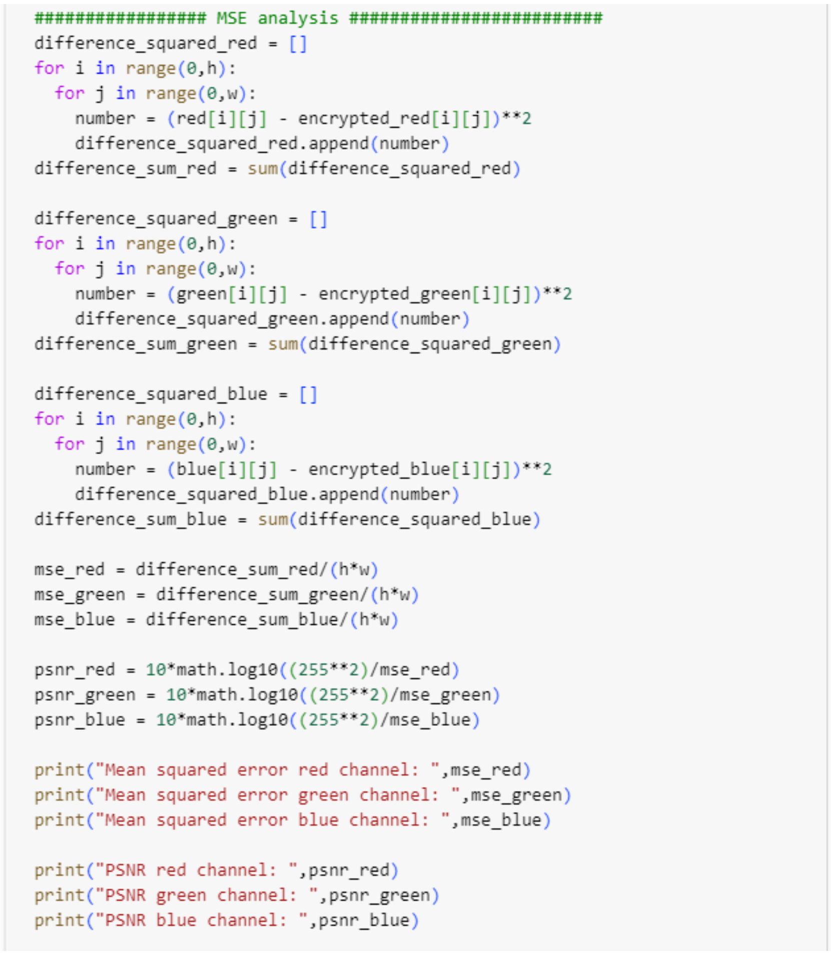
Result –





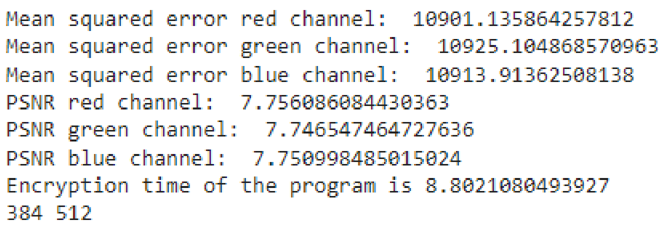


C) MSE analysis –



```
################# MSE analysis ########################
difference_squared_red = []
for i in range(0,h):
  for j in range(0,w):
    number = (red[i][j] - encrypted_red[i][j])**2
    difference_squared_red.append(number)
difference_sum_red = sum(difference_squared_red)

difference_squared_green = []
for i in range(0,h):
  for j in range(0,w):
    number = (green[i][j] - encrypted_green[i][j])**2
    difference_squared_green.append(number)
difference_sum_green = sum(difference_squared_green)

difference_squared_blue = []
for i in range(0,h):
  for j in range(0,w):
    number = (blue[i][j] - encrypted_blue[i][j])**2
    difference_squared_blue.append(number)
difference_sum_blue = sum(difference_squared_blue)

mse_red = difference_sum_red/(h*w)
mse_green = difference_sum_green/(h*w)
mse_blue = difference_sum_blue/(h*w)

psnr_red = 10*math.log10((255**2)/mse_red)
psnr_green = 10*math.log10((255**2)/mse_green)
psnr_blue = 10*math.log10((255**2)/mse_blue)

print("Mean squared error red channel: ",mse_red)
print("Mean squared error green channel: ",mse_green)
print("Mean squared error blue channel: ",mse_blue)

print("PSNR red channel: ",psnr_red)
print("PSNR green channel: ",psnr_green)
print("PSNR blue channel: ",psnr_blue)
```

Result –



```
Mean squared error red channel:  10901.135864257812
Mean squared error green channel:  10925.104868570963
Mean squared error blue channel:  10913.91362508138
PSNR red channel:  7.756086084430363
PSNR green channel:  7.746547464727636
PSNR blue channel:  7.750998485015024
Encryption time of the program is 8.8021080493927
384 512
```

Similar to above encryption and decryption processes are applied on other Algorithms -

- ✓ Tinker bell Algorithm + Duffing Algorithm

- ✓ Tinker bell Algorithm + Henon Algorithm

Found the Best resultant combination of algorithm which is

- ✓ Duffing Algorithm + Henon Algorithm

Which is explained above manual configuration.

References

1. Python (2019). Python For Beginners. [online] Python.org. Available at: https://www.python.org/about/gettingstarted/.

2. colab.research.google.com. (n.d.). Google Colaboratory. [online] Available at: https://colab.research.google.com/github/cs231n/cs231n.github.io/blob/master/python-colab.ipynb.