



**School of Computer Science Engineering**

# **Image Encryption and Decryption using Rubik's Cube Algorithm**

**By**

**Kulvir Singh (19BCE2074)**

**Project Report  
on  
CSE4019 – Image Processing**

**Fall Semester 2021-22**

**Submitted to  
Faculty: Dr. Sureshkumar N  
Date: 08-12-2021  
Slot : F1+TF1**

<b>CONTENTS</b>	<b>PAGE No.</b>
Abstract	3
Introduction	4
Literature Survey	6
Problem Statement	9
Research Framework	9
Modules	12
Code	14
Result and Analysis	21
Conclusion and Future Scope	26
References	27

# 1.Abstract

In today's world, almost 80% of our lives have been digitized and in the near future everyone and everything will be connected to a digital system of the internet and its subsidiaries. This paves the way for a huge security concern in the form of privacy of information and data. To evade such privacy concerns various countermeasures, need to be incorporated. Image encryption and decryption is just an example of one such countermeasure. In recent times several algorithms have been put forth based on chaotic systems for encryption and decryption, however they offer very limited security. In this project a new encryption technique is discussed which will use the Rubik's Cube Algorithm. The algorithm will be applied on coloured images and the pixel values of red, blue and green will be in action. The pixel values will be used in the algorithm to provide encryption by scrambling them using the XOR operation. The decryption of the image will be the reverse of the encryption process with the help of the keys produced in the previous phase. On successful completion of the project, one will be able to perform full encryption and decryption.

The user can input any image coloured or black-white into the algorithm. The program will then encrypt the image and scramble the pixel value using the mentioned algorithm and produce two security private keys and the encrypted image. The encrypted image can be decrypted by providing the keys and the encrypted image into the program and the final result would be the retrieval of the original input image.

Keywords:

Rubik's Cube, XOR, chaotic systems, noise utilization, pixel scrambling, asymmetric encryption, symmetric encryption

## 2.Introduction

Since the dawn of the computer age, security has been a primary goal of man. The advancements in the field of security have been duly noted and has been seen in the form of various security mechanisms and algorithms being developed. Cryptography has always played a major role and the combination of numbers and operations has proved to be an amazing tool to encapsulate user data. There are various cryptographic paradigms that involve ciphers based on mathematical tools. Since it all comes down to playing with numbers, the same tools have been used by researchers to provide encryption of images by manipulating pixel values. The pixel values of images are just numbers which can be scrambled, jumbled to make a new sequence and in turn a new distorted image. This project has a particular interest in providing image security using image processing techniques mixed with a security algorithm.

There are various techniques, methods and schemes revolving around image encryption and decryption. Waness et al. [8] has presented three particular methods of image encryption that are - symmetric, asymmetric and chaotic systems. Out of the three symmetric systems are rarely used, hence a deep study of chaotic system techniques and asymmetric algorithms have been made before selecting the choice of algorithm to be used in this project. Some valuable and resourceful information on chaotic systems is given below. [4][5]

One of the chaotic system techniques dealt with use of a generalized logistic map for real time image processing [5]. Shah et al. [5] presents a scheme that makes use of encryption with an efficient permutation based technique based on a modular logistic map to bring down the size of the chaotic value vector, required to permute a real-time image. The chaotic key sequence used in this technique changes the pixel values of a grayscale image. This optimises the chaotic key generation but is only extended on a grayscale image. Another chaotic system based encryption deals with double spiral scans and chaotic maps [4]. Tang et al. [4] presents the Double Spiral Scans which take a random scan of the image matrix and jumble those particular locations it scans twice. This results in randomization of the image and hence makes it distorted and encrypted. However if the entire image is based on the same pixel value then there will be no encryption despite the double scans.

Another technique is by using basic image processing concepts [7][9][10]. They make use of XOR operations along with image matrix arithmetic operations. Ahmad et al. [9] makes use of XOR and the formation of the orthogonal matrix and skew tent map using image pixels for providing a secure cipher image. Astuti et al. [10] deals with steganography and gives a vivid description on steganography. It also uses bit plane slicing along with XOR operation to work on the image and provide security. Another technique discussed is really interesting as it revolves around using destructive techniques on images to provide for security [7]. Mivule et al. [7] presents a scheme that will add noise to the image matrix and perform certain arithmetic operations on the pixels to change the original image. This technique however would not be beneficial for the retrieval process as the computation would be very difficult.

Another approach to encryption is one which is advanced and allows cloud features to come into play. Fu et al. [2] deals with the outsourcing of computation to a third party which provides image processing resources and thereby giving security. It deals with the application of an extended code base which helps other applications to function at the same computational complexity level without having to spend more on the computation of the image encryption algorithm. Another extension to this uses traditional image processing techniques combined with a homomorphic secret key to work on floating point numbers and provide an encrypted image [3].

Also, multiple images can be encrypted at once using ghost imaging. This deals with a multiple image encryption technique in which a group of plain images are encrypted. Each plain image is encrypted into an intensity vector by using the computational ghost vectoring scheme. Then all the vectors are superimposed on each other to result in a single ciphertext image [6].

Finally after a thorough examination and survey of all possible techniques, a new algorithm became the inspiration for this project's encryption and decryption mechanism. It deals with the Rubik's Cube algorithm [1]. This is a novel algorithm which can encrypt an image without any chaotic means and uses the asymmetric encryption-decryption ideology combined with XOR operations and certain arithmetic operations. The algorithm suggested only applies to gray scale images but with the help from multiple image encryption techniques as seen above my project

aims to extend it to coloured images. The coloured images can be divided into three separate planes of red, blue and green pixel values. These pixel values will form three different image matrices which will be encrypted separately and then superimposed together to get the final single encrypted image as the output. For decryption, the reverse of the encryption process will be followed which means dividing the encrypted image into red, blue and green pixel matrices, performing decryption and then superimposing to get the original image back.

### **3.Literature Review**

- **Efficient image encryption scheme based on generalized logistic map for real time image processing**

The paper discusses an efficient encryption algorithm which has been tested with real-time images. The scheme makes use of encryption with an efficient permutation based technique based on a modular logistic map to bring down the size of the chaotic value vector, required to permute a real-time image. The chaotic key sequence used in this technique changes the pixel values of a grayscale image. This optimises the chaotic key generation but is only extended on a grayscale image.

- **Image Encryption with Double Spiral Scans and Chaotic Maps**

This paper discusses a chaotic map based encryption technique which uses two scans of the image matrix. The Double Spiral Scans take a random scan of the image matrix and jumble those particular locations it scans. This results in randomization of the image and hence makes it distorted and encrypted. This process is repeated twice hence the name double spiral scan. However if the entire image is based on the same pixel value then there will be no encryption despite the double scans.

- **Fully Homomorphic Image Processing**

This paper deals with the outsourcing of computation to a third party which provides image processing resources and thereby giving security. It deals with the application of an extended code base which helps other applications to function at the same computational complexity level without having to spend more on the computation of the image encryption algorithm. This paper also shows the fully homomorphic image processing techniques and its drawbacks.

- **An efficient secret key homomorphic encryption used in image processing service**

This paper has explained the homomorphic encryption technique for images. This technique uses traditional image processing techniques combined with a homomorphic secret key to work on floating point numbers and provide an encrypted image. The technique is quite advanced and complex and can be hosted to provide security on cloud platforms. The paper explains the use of coloured images and the manipulation of the colour pixel values.

- **A secure image encryption algorithm based on Rubik's cube principle**

This paper explains the drawbacks of traditional encryption techniques for image processing. It uses the Rubik's cube algorithm and implements image processing methods to get an encrypted image. The paper only deals with a single grayscale image matrix and hence cannot be extended to coloured images.

- **Multiple-image encryption based on computational ghost imaging.**

This paper deals with a multiple image encryption technique in which a group of plain images are encrypted. Each plain image is encrypted into an intensity vector by using the computational ghost vectoring scheme. Then all the vectors are superimposed on each other to result in a single ciphertext image. A similar model work was described and gives a great insight on how to work with multiple images.

- **Utilizing Noise Addition for Data Privacy, an Overview**

This paper gives a very unique idea of encrypting an image by spoiling the image. This scheme will add noise to the image matrix and perform certain arithmetic operations on the pixels to change the original image. This technique however would not be beneficial for the retrieval process as the computation would be very difficult.

- **A Survey on the Image Encryption Methods.**

This paper deals with various image encryption techniques. It gives a comparative analysis of all the various systems of encryption including symmetric, asymmetric and chaotic based. The paper provides an overview and a good distinctive study of all these techniques.

- **A novel image encryption scheme based on orthogonal matrix, skew tent map, and XOR operation**

This paper deals with the image encryption based on the XOR operation and other image arithmetic operations. The paper explains the use of XOR and the formation of the orthogonal matrix and skew tent map using image pixels for providing a secure cipher image.

- **Simple and secure image steganography using LSB and triple XOR operation on MSB**

This paper deals with steganography and gives a vivid description on steganography. It also uses bit plane slicing along with XOR operation to work on the image and provide



## 4. Problem Statement

Any image when provided as an input to the project should result in the generation of a new image which is distorted, unrecognisable and has minimal resemblance to the input image. Also, a file containing keys should be generated. When the distorted image is input in the project with the set of keys, the original image should be generated.

## 5. Research Framework

The project is supported by the following research architecture which is discussed below. All the modules of the project operate on the basis of simple image arithmetic and logical operations. These operations are very simple and easy to use. They can operate in very minimal configuration software and thus the time and space complexity to use these operators would be very low. Hence the entire project is optimized time and space wise and easy to use. First the arithmetic operations used shall be discussed.

The first operation used is the arithmetic summation operation(see Fig. 1). This operation adds the pixel values and stores it in a resultant matrix. The pixel values are added in the following way: the (i,j)th pixel value of matrix P1 is added with the (i,j)th pixel value of matrix P2 and then the sum of the two is stored in the same (i,j)th location of the sum matrix Q.

$$Q(i, j) = P_1(i, j) + P_2(i, j)$$

$$Q(i, j) = P_1(i, j) + C$$

**Fig. 1.** Arithmetic Sum Operation

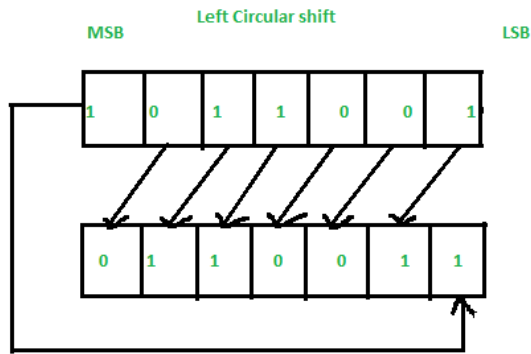
Another operation used is the modulus operation. According to the project, each pixel of an image matrix is divided with a constant value and the remainder of that division is stored in a new matrix at the same location of the earlier pixel. In mathematical notation, one can say that

$$Q(i,j) = P1(i,j)\%P2$$

$$Q(i,j) = P1(i,j)\%C$$

Other operations include logical and position shifting operations.

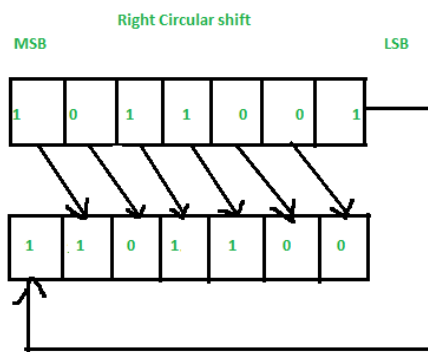
Left Circular Shift (see Fig. 2) is used on the image matrix. In left circular shift the pixels of a row of an image matrix are shifted by one position to the left without the loss of any pixel value



**Fig. 2.** Left Circular Shift Operation

Mathematically, for each row of the image matrix, the 0th pixel is placed at the mth location while all the other pixel values (1 to m) are placed at a location one less than their current position (0 to m-1).

Right Circular Shift (see Fig. 3) is used on the image matrix. In right circular shift the pixels of a row of an image matrix are shifted by one position to the right without the loss of any pixel value



**Fig. 3.** Right Circular Shift Operation

Mathematically, for each row of the image matrix, the  $m$ th pixel is placed at the 0th location while all the other pixel values (0 to  $m-1$ ) are placed at a location one more than their current position (1 to  $m$ ).

Downshift is used on the image matrix. In downshift the pixels of a column of an image matrix are shifted by one position below without the loss of any pixel value. It can be interpreted as a vertical right circular shift.

Mathematically, for each column of the image matrix, the  $n$ th pixel is placed at the 0th location while all the other pixel values (0 to  $n-1$ ) are placed at a location one more than their current position (1 to  $n$ ).

Upshift is used on the image matrix. In upshift the pixels of a column of an image matrix are shifted by one position above without the loss of any pixel value. It can be interpreted as a vertical left circular shift.

Mathematically, for each column of the image matrix, the 0th pixel is placed at the  $n$ th location while all the other pixel values (1 to  $n$ ) are placed at a location one more than their current position (0 to  $n-1$ ).

Lastly an XOR operation is used on the image matrix. This is a logical operation which will do the following on an image matrix:

The XOR function is only true if just one (and only one) of the input values is true, and false otherwise. XOR stands for eXclusive OR. As can be seen, the output values of XNOR are simply the inverse of the corresponding output values of XOR.

The XOR (and similarly the XNOR) (see Fig. 4) operator typically takes two binary or gray level images as input, and outputs a third image whose pixel values are just those of the first image, XORed with the corresponding pixels from the second. A variation of this operator takes a single input image and XORs each pixel with a specified constant value in order to produce the output.

A	B	Q
0	0	0
0	1	1
1	0	1
1	1	0

**XOR**

A	B	Q
0	0	1
0	1	0
1	0	0
1	1	1

**XNOR**

**Fig. 4.** Truth table for XOR and XNOR.

## 6. Modules

The entire project is divided into 2 major modules namely - Encryption Module and Decryption Module along with the pre-processing and post-processing. Furthermore a utility function module is also created to assist the major modules functioning.

### Image Pre-processing

In this module, a colored image is taken as an input. The image is loaded into the pre-processing module and the following changes happen. The colored image is divided into the primary color matrices ie the red, blue and green matrices. The pixel values of all the 3 matrices are obtained. Next, the encryption keys are set and stored according to the Rubik's Cube algorithm.

### Image Encryption Module

In this module, the input image would be encrypted after pre-processing using the techniques mentioned in the research framework. The sum of the rows of all the red, blue and green values

is calculated one by one. The sum is then stored separately for the three values after performing mod 2 operation. After performing that, a check is made if the modulus value is 0 or not. If the value is 0 then a right circular shift is made a certain number of times according to the keys generated in the pre-processing module else left circular shift. The same process is repeated for the green and blue values also. After processing the rows, we process the columns. The sum operation is applied and the modulus 2 values are stored for the red, blue and green matrix. After performing that, a check is made if the modulus value is 0 or not. If the value is 0 then a upshift is made a certain number of times according to the keys generated in the pre-processing module else downshift. After upshift and downshift and circular shifting, XOR operation is performed on the resultant red, blue and green matrix according to the key values.

### **Image Decryption Module**

After following the image pre-processing module another time, the reverse of the encryption process is followed. The XOR operation is performed on the red, blue and green matrix obtained from the pre-processed stage. The XOR is performed according to the keys entered by the user during the decryption input phase. Now we process the columns of the modified red, blue and green matrices obtained. The sum operation is applied and the modulus 2 values are stored for the red, blue and green matrix. After performing that, a check is made if the modulus value is 0 or not. If the value is 0 then a upshift is made a certain number of times according to the keys generated in the pre-processing module else downshift. After processing the columns we process the rows. The sum of the rows of all the red, blue and green values is calculated one by one. The sum is then stored separately for the three values after performing mod 2 operation. After performing that, a check is made if the modulus value is 0 or not. If the value is 0 then a right circular shift is made a certain number of times according to the keys generated in the pre-processing module else left circular shift. The same process is repeated for the green and blue values also. Finally the three matrices obtained undergo post processing to retrieve the original form of the image.

## Image post-processing

After generating the modified pixel values for the red, blue and green image matrix, these are superimposed and combined together to form an encrypted image. The encrypted image is then stored and is free for use. Apart from this, keys are generated for decrypting the same in a txt file.

## 7. Code

### Encrypt.py

```
import os
from PIL import Image
from random import randint
import numpy
import sys
def upshift(a,index,n):
    col = []
    for j in range(len(a)):
        col.append(a[j][index])
    shiftCol = numpy.roll(col,-n)
    for i in range(len(a)):
        for j in range(len(a[0])):
            if(j==index):
                a[i][j] = shiftCol[i]
def downshift(a,index,n):
    col = []
    for j in range(len(a)):
        col.append(a[j][index])
    shiftCol = numpy.roll(col,n)
    for i in range(len(a)):
        for j in range(len(a[0])):
            if(j==index):
                a[i][j] = shiftCol[i]
def rotate180(n):
    bits = "{0:b}".format(n)
```

```

    return int(bits[::-1], 2)

#loading image from a directory
im =
Image.open(os.path.join('C:\\Users\\kulvir\\Desktop\\RubikAlg\\', 'i
mage.PNG'))

pix = im.load()#converting image to pixels as python object

#Obtaining the RGB matrices
r = []
g = []
b = []
for i in range(im.size[0]):
    r.append([])
    g.append([])
    b.append([])
    for j in range(im.size[1]):
        rgbPerPixel = pix[i,j]
        r[i].append(rgbPerPixel[0])
        g[i].append(rgbPerPixel[1])
        b[i].append(rgbPerPixel[2])

# M x N image matrix
m = im.size[0] #rows
n = im.size[1] #columns

# Vectors Kr and Kc
alpha = 8
Kr = [randint(0,pow(2,alpha)-1) for i in range(m)]
Kc = [randint(0,pow(2,alpha)-1) for i in range(n)]

#maximum number of iterations
ITER_MAX = 1

print('Vector Kr : ', Kr)
print('Vector Kc : ', Kc)

#key for encryption written into the file keys.txt
f = open('keys.txt','w+')
f.write('Vector Kr :\n')

```

```

for a in Kr:
    f.write(str(a) + '\n')
f.write('Vector Kc :\n')
for a in Kc:
    f.write(str(a) + '\n')
f.write('ITER_MAX :\n')
f.write(str(ITER_MAX) + '\n')

for iterations in range(ITER_MAX):
    # For each row
    for i in range(m):
        rTotalSum = sum(r[i]) #sum of each array present in r[][]
        gTotalSum = sum(g[i])
        bTotalSum = sum(b[i])
        #modulo of sum of each r,g,b
        rModulus = rTotalSum % 2
        gModulus = gTotalSum % 2
        bModulus = bTotalSum % 2

        if(rModulus==0):
            #right circular shift according to Kr
            r[i] = numpy.roll(r[i],Kr[i])
        else:
            #left circular shift according to Kr
            r[i] = numpy.roll(r[i],-Kr[i])
        if(gModulus==0):
            g[i] = numpy.roll(g[i],Kr[i])
        else:
            g[i] = numpy.roll(g[i],-Kr[i])
        if(bModulus==0):
            b[i] = numpy.roll(b[i],Kr[i])
        else:
            b[i] = numpy.roll(b[i],-Kr[i])
    # For each column
    for i in range(n):
        rTotalSum = 0
        gTotalSum = 0
        bTotalSum = 0
        for j in range(m):
            rTotalSum += r[j][i]
            gTotalSum += g[j][i]
            bTotalSum += b[j][i]

```



```

rModulus = rTotalSum % 2
gModulus = gTotalSum % 2
bModulus = bTotalSum % 2
if (rModulus == 0):
    upshift(r, i, Kc[i])
else:
    downshift(r, i, Kc[i])
if (gModulus == 0):
    upshift(g, i, Kc[i])
else:
    downshift(g, i, Kc[i])
if (bModulus == 0):
    upshift(b, i, Kc[i])
else:
    downshift(b, i, Kc[i])

# For each row
for i in range(m):
    for j in range(n):
        if (i % 2 == 1):
            r[i][j] = r[i][j] ^ Kc[j]
            g[i][j] = g[i][j] ^ Kc[j]
            b[i][j] = b[i][j] ^ Kc[j]
        else:
            r[i][j] = r[i][j] ^ rotate180(Kc[j])
            g[i][j] = g[i][j] ^ rotate180(Kc[j])
            b[i][j] = b[i][j] ^ rotate180(Kc[j])

# For each column
for j in range(n):
    for i in range(m):
        if (j % 2 == 0):
            r[i][j] = r[i][j] ^ Kr[i]
            g[i][j] = g[i][j] ^ Kr[i]
            b[i][j] = b[i][j] ^ Kr[i]
        else:
            r[i][j] = r[i][j] ^ rotate180(Kr[i])
            g[i][j] = g[i][j] ^ rotate180(Kr[i])
            b[i][j] = b[i][j] ^ rotate180(Kr[i])

for i in range(m):

```

```

for j in range(n):
    pix[i,j] = (r[i][j],g[i][j],b[i][j])
im.save('C:\\Users\\kulvir\\Desktop\\RubikAlg\\encrypted.PNG')
print("Success")

```

### decrypt.py:-

```

import os
from PIL import Image
import numpy
import sys
#to load encrypted image
im =
Image.open(os.path.join('C:\\Users\\kulvir\\Desktop\\RubikAlg\\', 'e
ncrypted.PNG'))
pix = im.load() #pixels loaded to pix array

def upshift(a,index,n):
    col = []
    for j in range(len(a)):
        col.append(a[j][index])
    shiftCol = numpy.roll(col,-n)
    for i in range(len(a)):
        for j in range(len(a[0])):
            if(j==index):
                a[i][j] = shiftCol[i]
def downshift(a,index,n):
    col = []
    for j in range(len(a)):
        col.append(a[j][index])
    shiftCol = numpy.roll(col, n)
    for i in range(len(a)):
        for j in range(len(a[0])):
            if (j == index):
                a[i][j] = shiftCol[i]
def rotate180(n):
    bits = "{0:b}".format(n)
    return int(bits[::-1], 2)
#Obtaining the RGB matrices
r = []
g = []
b = []

```

```

#obtainge rgb values from pix
for i in range(im.size[0]):
    r.append([])
    g.append([])
    b.append([])
    for j in range(im.size[1]):
        rgbPerPixel = pix[i, j]
        r[i].append(rgbPerPixel[0])
        g[i].append(rgbPerPixel[1])
        b[i].append(rgbPerPixel[2])
m = im.size[0]
n = im.size[1]
#key input from user
Kr = []
Kc = []
print('Enter value of Kr')
for i in range(m):
    Kr.append(int(input()))
print('Enter value of Kc')
for i in range(n):
    Kc.append(int(input()))
print('Enter value of ITER_MAX')
ITER_MAX = int(input())

for iterations in range(ITER_MAX):
    # For each column
    for j in range(n):
        for i in range(m):
            if (j % 2 == 0):
                r[i][j] = r[i][j] ^ Kr[i]
                g[i][j] = g[i][j] ^ Kr[i]
                b[i][j] = b[i][j] ^ Kr[i]
            else:
                r[i][j] = r[i][j] ^ rotate180(Kr[i])
                g[i][j] = g[i][j] ^ rotate180(Kr[i])
                b[i][j] = b[i][j] ^ rotate180(Kr[i])
    # For each row
    for i in range(m):
        for j in range(n):
            if (i % 2 == 1):
                r[i][j] = r[i][j] ^ Kc[j]
                g[i][j] = g[i][j] ^ Kc[j]

```

```

        b[i][j] = b[i][j] ^ Kc[j]
    else:
        r[i][j] = r[i][j] ^ rotate180(Kc[j])
        g[i][j] = g[i][j] ^ rotate180(Kc[j])
        b[i][j] = b[i][j] ^ rotate180(Kc[j])
# For each column
for i in range(n):
    rTotalSum = 0
    gTotalSum = 0
    bTotalSum = 0
    for j in range(m):
        rTotalSum += r[j][i]
        gTotalSum += g[j][i]
        bTotalSum += b[j][i]
    rModulus = rTotalSum % 2
    gModulus = gTotalSum % 2
    bModulus = bTotalSum % 2
    if (rModulus == 0):
        downshift(r, i, Kc[i])
    else:
        upshift(r, i, Kc[i])
    if (gModulus == 0):
        downshift(g, i, Kc[i])
    else:
        upshift(g, i, Kc[i])
    if (bModulus == 0):
        downshift(b, i, Kc[i])
    else:
        upshift(b, i, Kc[i])
# For each row
for i in range(m):
    rTotalSum = sum(r[i])
    gTotalSum = sum(g[i])
    bTotalSum = sum(b[i])
    rModulus = rTotalSum % 2
    gModulus = gTotalSum % 2
    bModulus = bTotalSum % 2
    if (rModulus == 0):
        r[i] = numpy.roll(r[i], -Kr[i])
    else:
        r[i] = numpy.roll(r[i], Kr[i])
    if (gModulus == 0):

```

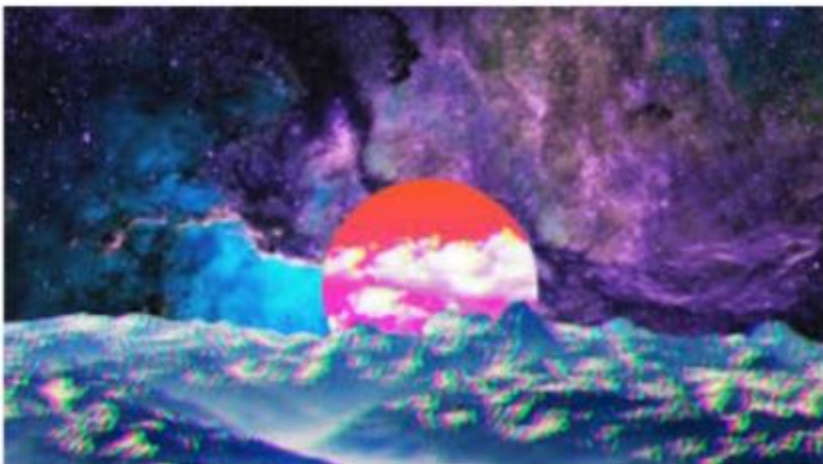
```

        g[i] = numpy.roll(g[i], -Kr[i])
    else:
        g[i] = numpy.roll(g[i], Kr[i])
    if (bModulus == 0):
        b[i] = numpy.roll(b[i], -Kr[i])
    else:
        b[i] = numpy.roll(b[i], Kr[i])
for i in range(m):
    for j in range(n):
        pix[i, j] = (r[i][j], g[i][j], b[i][j])
im.save('C:\\Users\\kulvir\\Desktop\\RubikAlg\\original.PNG')
print("Success")

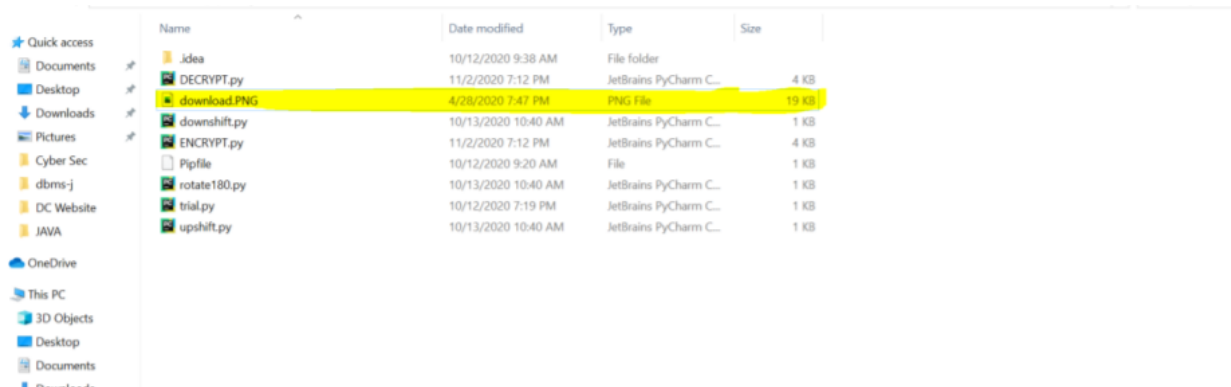
```

## 8. Results and Analysis

The following image is encrypted. The image file is under the name of 'download.PNG'.



The image to be encrypted is to be saved in the project directory. The image file is named 'download.PNG' and is highlighted below.

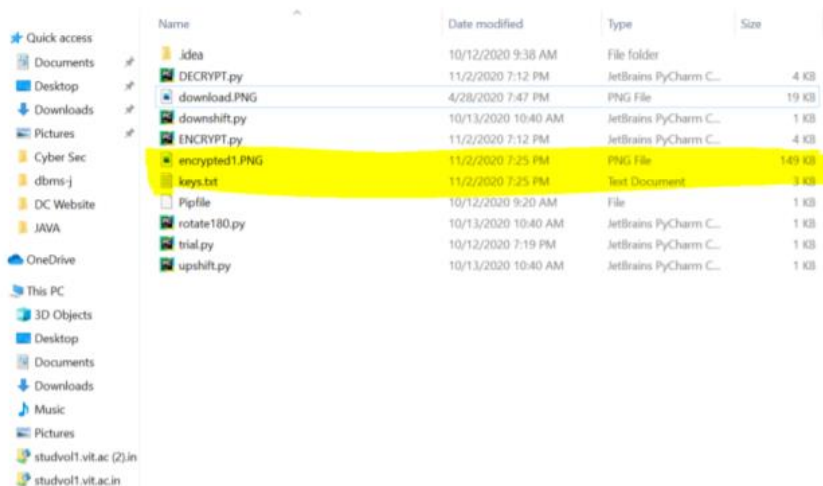


Name	Date modified	Type	Size
.idea	10/12/2020 9:38 AM	File folder	
DECRYPT.py	11/2/2020 7:12 PM	JetBrains PyCharm C...	4 KB
download.PNG	4/28/2020 7:47 PM	PNG File	19 KB
downshift.py	10/13/2020 10:40 AM	JetBrains PyCharm C...	1 KB
ENCRYPT.py	11/2/2020 7:12 PM	JetBrains PyCharm C...	4 KB
Pipfile	10/12/2020 9:20 AM	File	1 KB
rotate180.py	10/13/2020 10:40 AM	JetBrains PyCharm C...	1 KB
trial.py	10/12/2020 7:19 PM	JetBrains PyCharm C...	1 KB
upshift.py	10/13/2020 10:40 AM	JetBrains PyCharm C...	1 KB

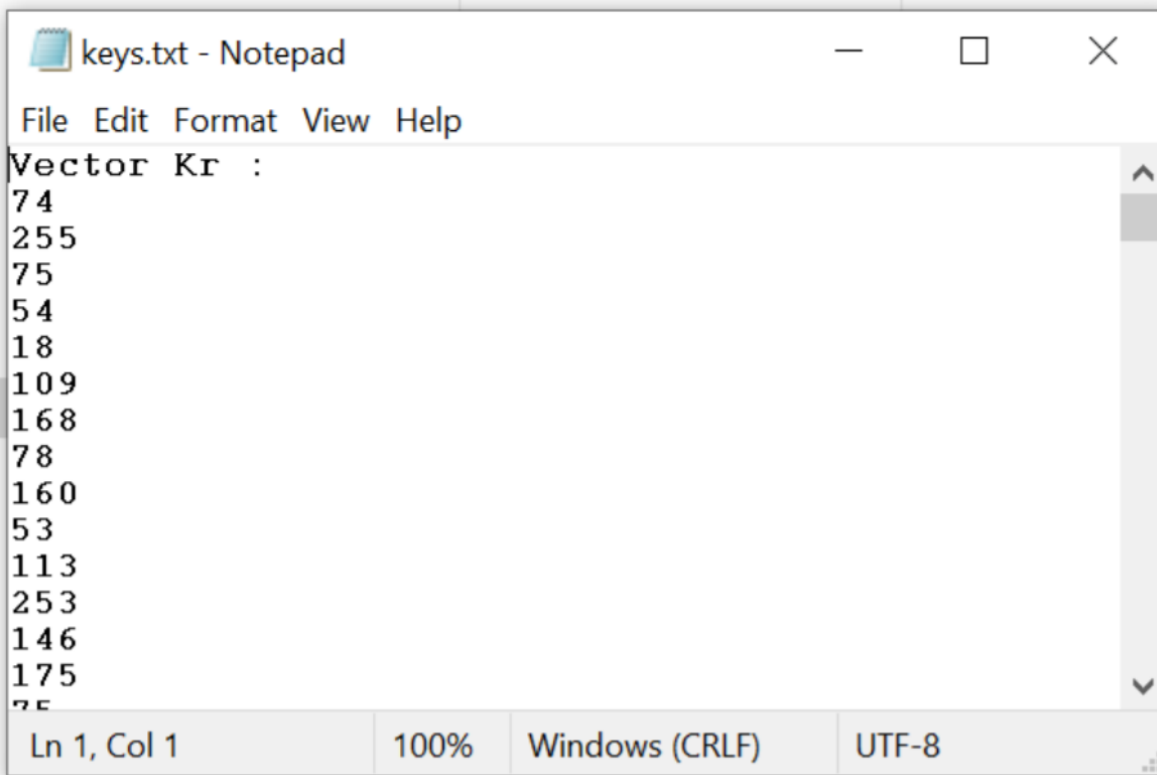
On running the encrypt.py file in the terminal, the keys are displayed and the encrypted image along with the keys.txt file is created

```
C:\Users\kulvir\Desktop\RubikAlg>python ENCRYPT.py
Vector Kr : [125, 40, 30, 255, 64, 160, 65, 190, 149, 131, 156, 208, 191, 5, 133, 184, 53, 226, 175, 157, 243, 67, 48, 142, 1
45, 169, 24, 203, 192, 197, 173, 12, 7, 49, 89, 166, 155, 201, 114, 55, 122, 3, 254, 122, 202, 25, 243, 188, 159, 2, 163, 135,
125, 61, 114, 121, 40, 246, 15, 135, 97, 46, 96, 163, 24, 166, 205, 130, 99, 252, 149, 201, 136, 251, 166, 182, 98, 195, 212,
22, 253, 250, 146, 175, 175, 103, 41, 227, 62, 150, 63, 75, 0, 199, 83, 48, 72, 160, 178, 228, 154, 16, 103, 71, 102, 9, 245,
118, 203, 233, 190, 107, 166, 71, 10, 252, 90, 118, 23, 247, 171, 202, 194, 37, 53, 220, 176, 96, 42, 19, 5, 223, 183, 23, 60
, 179, 233, 242, 135, 191, 159, 173, 97, 241, 79, 39, 37, 148, 145, 213, 79, 217, 231, 227, 237, 2, 114, 224, 17, 72, 199, 89,
70, 188, 12, 57, 102, 76, 243, 251, 59, 88, 224, 133, 68, 20, 92, 32, 221, 155, 253, 47, 72, 173, 68, 213, 187, 203, 112, 103
, 238, 184, 21, 183, 35, 192, 40, 133, 7, 196, 169, 121, 50, 179, 12, 41, 168, 24, 32, 243, 77, 124, 31, 230, 156, 7, 72, 212,
24, 146, 129, 230, 252, 247, 12, 103, 98, 128, 188, 179, 246, 233, 13, 30, 23, 196, 95, 192, 204, 238, 79, 53, 155, 208, 175,
99, 59, 194, 55, 248, 106, 72, 71, 174, 216, 170, 145, 77, 254, 69, 25, 11, 24, 190, 170, 219, 40, 237, 217, 74, 50, 20, 25,
46, 215, 244, 201, 47, 232, 83, 90, 198, 224, 183, 82, 90, 250, 191, 246, 77, 106, 57, 148, 126, 31, 96, 65, 76, 227, 18]
Vector Kc : [30, 95, 206, 24, 105, 15, 11, 244, 116, 73, 44, 201, 176, 30, 160, 160, 99, 221, 96, 126, 13, 149, 115, 156, 15,
47, 43, 75, 112, 206, 235, 76, 148, 251, 228, 203, 54, 140, 89, 182, 164, 218, 42, 249, 229, 95, 17, 100, 70, 6, 240, 187, 29
, 199, 59, 135, 121, 28, 20, 195, 0, 245, 125, 5, 201, 33, 207, 44, 148, 149, 13, 154, 134, 1, 20, 134, 62, 14, 158, 68, 4, 2,
126, 180, 127, 155, 127, 113, 166, 2, 211, 188, 202, 44, 128, 35, 42, 252, 79, 236, 211, 230, 225, 205, 50, 19, 209, 45, 185,
58, 197, 255, 245, 188, 144, 250, 232, 115, 108, 127, 160, 70, 195, 75, 240, 84, 225, 70, 124, 225, 184, 9, 169, 63, 182, 170
, 32, 22, 95, 39, 18, 254, 49, 114, 206, 64, 197, 246, 63, 223, 228, 91, 254, 22, 126, 95, 98, 242, 250, 107, 190, 33, 214, 14
2, 89, 253, 232, 140]
Success
```

The encrypted image is generated in the encrypted1.png file and the keys are stored in the keys.txt file as highlighted below.



The keys.txt file containing the keys required for decryption.



The 'download.PNG' is encrypted successfully and the encrypted image('encrypted1.PNG') is shown below.



On running the decrypt.py file in terminal to decrypt the encrypted image, we first need to copy the keys from the keys.txt file and enter it into the terminal as shown below.

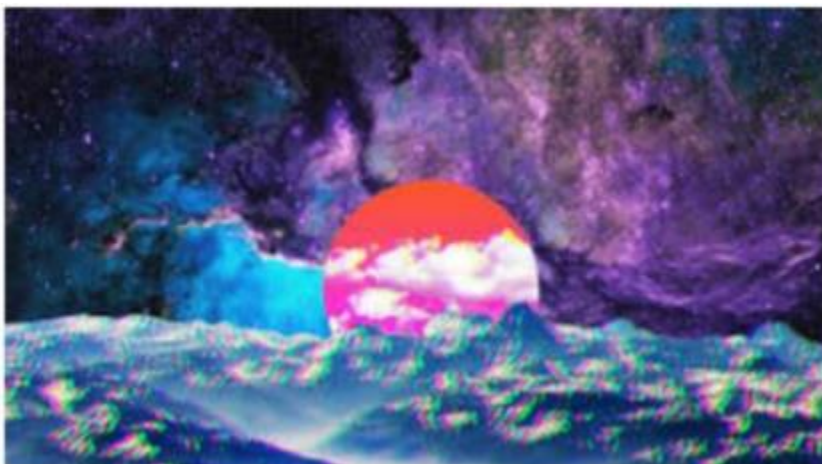
```
117
121
184
231
49
135
183
24
193
210
24
27
180
112
136
118
94
74
73
225
91
Enter value of ITER_MAX
1
Success
```

After entering all the keys, the code gets executed which generates the decrypted image which is stored under the same folder with the name 'original.PNG'



★ Quick access				
Documents	.idea	10/12/2020 9:38 AM	File folder	
Desktop	DECRYPT.py	11/2/2020 7:12 PM	JetBrains PyCharm C...	4 KB
Downloads	download.PNG	4/28/2020 7:47 PM	PNG File	19 KB
Pictures	downshift.py	10/13/2020 10:40 AM	JetBrains PyCharm C...	1 KB
Cyber Sec	ENCRYPT.py	11/2/2020 7:12 PM	JetBrains PyCharm C...	4 KB
dbms-j	encrypted1.PNG	11/2/2020 7:25 PM	PNG File	149 KB
DC Website	keys.txt	11/2/2020 7:25 PM	Text Document	3 KB
JAVA	original.PNG	11/2/2020 7:30 PM	PNG File	113 KB
OneDrive	Pipfile	10/12/2020 9:20 AM	File	1 KB
This PC	rotate180.py	10/13/2020 10:40 AM	JetBrains PyCharm C...	1 KB
3D Objects	trial.py	10/12/2020 7:19 PM	JetBrains PyCharm C...	1 KB
Desktop	upshift.py	10/13/2020 10:40 AM	JetBrains PyCharm C...	1 KB
Documents				
Downloads				
Music				
Pictures				

The original.PNG is the same as the download.PNG image hence decryption is performed successfully.



## **9. Conclusion**

The proposed system and research framework shows that using the Rubik's Cube algorithm one can encrypt any coloured image using the red, blue and green pixel values of the image. By following the reverse procedure, the image can be decrypted. The proposed methodology consisted of simple operations based on arithmetic and image processing. This can enable the system to process in an optimized environment as the encryption and decryption process does not take much time and space complexity. The files and outputs generated do not have a significant space requirement as compared to the input image. Also, the encrypted image is very difficult to decrypt with Brute-Force mechanism hence a high security parameter is ensured.

## **10. Future Scope**

The proposed system can be developed on various platforms and the implemented across various applications. The process can be used for protecting files and information on servers. The project can be extended to protect data hosted on online platforms. The system can be scaled onto different compiling languages which are faster than python, to increase the speed of computing.

## 11. References

- [1] Loukhaoukha, K., Chouinard, J. Y., & Berdai, A. (2012). A secure image encryption algorithm based on Rubik's cube principle. *Journal of Electrical and Computer Engineering*, 2012.
- [2] Fu, W., Lin, R., & Inge, D. (2018). Fully Homomorphic Image Processing. *arXiv preprint arXiv:1810.03249*.
- [3] Yang, P., Gui, X., An, J., & Tian, F. (2017). An efficient secret key homomorphic encryption used in image processing service. *Security and Communication Networks*, 2017.
- [4] Tang, Z., Yang, Y., Xu, S., Yu, C., & Zhang, X. (2019). Image encryption with double spiral scans and chaotic maps. *Security and Communication Networks*, 2019.
- [5] Shah, A. A., Parah, S. A., Rashid, M., & Elhoseny, M. (2020). Efficient image encryption scheme based on generalized logistic map for real time image processing. *Journal of Real-Time Image Processing*, 17(6), 2139-2151.
- [6] Wu, J., Xie, Z., Liu, Z., Liu, W., Zhang, Y., & Liu, S. (2016). Multiple-image encryption based on computational ghost imaging. *Optics Communications*, 359, 38-43.
- [7] Mivule, K. (2013). Utilizing noise addition for data privacy, an overview. *arXiv preprint arXiv:1309.3958*.
- [8] Abd El Waness, A. K., & Eldib, M. M. (2020). A Survey on the Image Encryption Methods. *Journal of Computer Science and Information Systems*, 11(2 June 2020).
- [9] Ahmad, J., Khan, M. A., Ahmed, F., & Khan, J. S. (2018). A novel image encryption scheme based on orthogonal matrix, skew tent map, and XOR operation. *Neural Computing and Applications*, 30(12), 3847-3857.

[10] Astuti, Y. P., Rachmawanto, E. H., & Sari, C. A. (2018, March). Simple and secure image steganography using LSB and triple XOR operation on MSB. In 2018 International Conference on Information and Communications Technology (ICOIACT) (pp. 191-195). IEEE.