Image Encryption and Decryption using Rubik's Cube Algorithm

Name : Kulvir Singh
Register Number : 19BCE2074

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 softwares 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. 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.

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

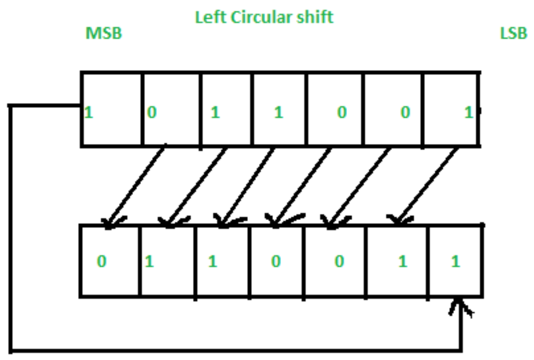
Another operation used is the modulus operation. According to the project, each pixel of an image matrix is divided with a constant value and the remainder of that division is stored in a new matrix at the same location of the earlier pixel. In mathematical notation, one can say that

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

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

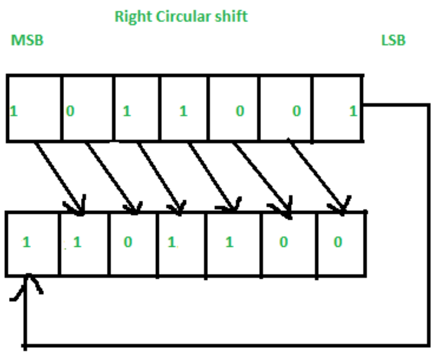
Other operations include logical and position shifting operations.

Left Circular Shift 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



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 ro m-1).

Right Circular Shift 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



Mathematically, for each row of the image matrix, the mth 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 ro 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.

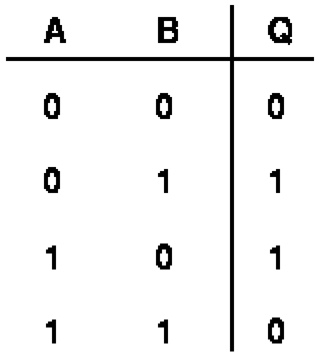
Mathematically, for each column of the image matrix, the nth pixel is placed at the 0th location while all the other pixel values (0 to n-1) are placed at a location one more than their current position(1 to n).

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 nth 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) 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

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.

Implementation

Pre-processing Code :

```
im =
Image.open(os.path.join('C:\\Users\\kulvir\\Desktop\\RubikAlg\\','ima
ge.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')
```

Encryption Code

```
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 shit 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])
```

Post-processing Code

```
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")
```

Sample Input :

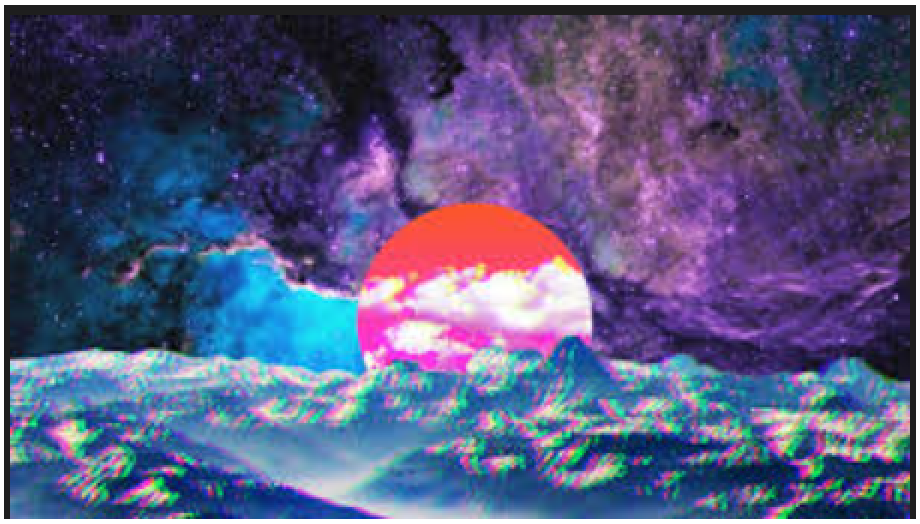
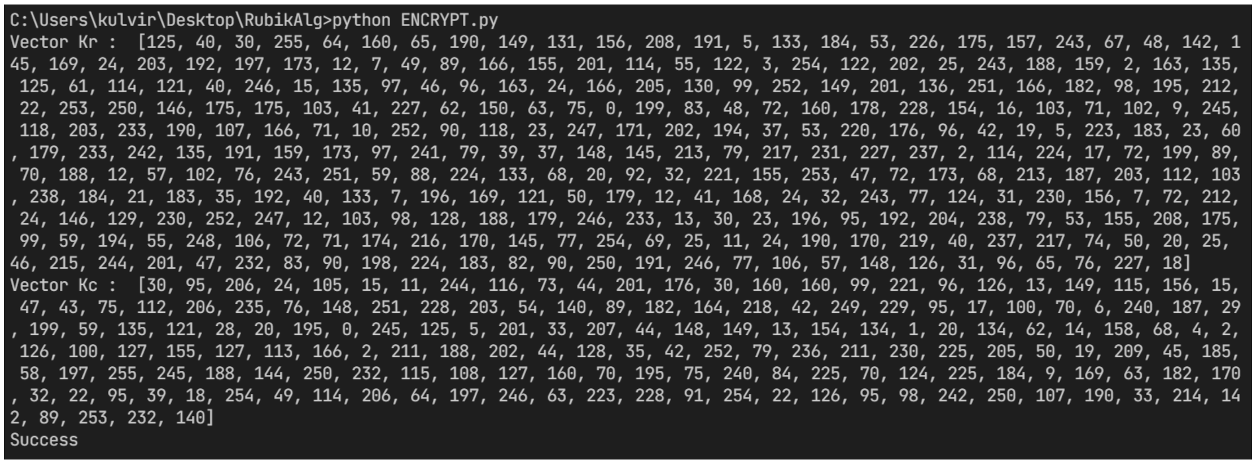


Image Processing :



```
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, 100, 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
```

Output Image (Encrypted Image) :