Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

FINAL REVIEW

VISUAL CRYPTOGRAPHY

Group Members

Anubhav Agarwal – 20BCE0714

Ruksana Tabassum – 20BCE2650

Manikonda Sri Pavani – 20BCT0283

Ritocheta Roy – 20BCI0332

Under the guidance of

Prof. Gayathri P

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

VIT, VELLORE

ABSTRACT

In the current technology-driven world, it is very essential for the users to maintain their data privacy, and to provide security and privacy, encryption decryption is a very important concept to protect user’s data from any unauthorised access. Cyber Security is essential for each individual to watch its records and advanced information to keep undesirable clients from getting to it. Image encryption is a useful technique for the protection of image content shared over internet. Different research publications on "Visual Cryptography" and related methods inspired us to create a better and more effective code and merge it into a working model. The project is based on the basic concept of visual cryptography. The solitary form of the visual cryptic issues recognises that the message is made up of a collection of Black and White pixels, each of which is dealt with separately.

Our fundamental thought process is that we will utilize a superior algorithm, that is, "Rubik's Cube algorithm" which is a picture encryption which fuses an RBG shading plan, “Hill cipher” which uses input from the user to create unique subkeys and “Affine cipher” which uses the subkeys from Hill cipher to perform various operations on the image. We have also decided to incorporate the use of MMI to further increase the security. We chose these algorithms because they are simple to grasp while still being incredibly safe. Instead of simply black and white photos, this combination of algorithms also allows us to encrypt and decrypt coloured images.

INTRODUCTION

To ensure the security of photographs, cryptography is applied. It is a method of encrypting an image with a key, resulting in a cypher image that is difficult to decipher. Future multimedia Internet applications will require image encryption. Password codes will most likely be replaced with biometric fingerprint pictures to identify individual users. However, such data will almost certainly be transmitted across a network.

When such images are transmitted over the internet, an eavesdropper may copy or reroute the data. By encrypting these photos, the content is still secure to some extent. Image encryption can also be used to keep personal information private. As a result, image encryption can be employed to mitigate these issues. Although encryption is sufficient to protect digital images and videos in some civil applications, these issues are considered when advanced encryption algorithms are specifically designed for sensitive digital images and videos, because their special features differ significantly from those of texts.

LITERATURE REVIEW

To ensure the security of photographs, cryptography is applied. It is a method of encrypting an image with a key, resulting in a cypher image that is difficult to decipher. Future multimedia Internet applications will require image encryption. Password codes will most likely be replaced with biometric fingerprint pictures to identify individual users. However, such data will almost certainly be transmitted across a network. When such images are transmitted over the internet, an eavesdropper may copy or reroute the data. By encrypting these photos, the content is still secure to some extent. Image encryption can also be used to keep personal information private. As a result, image encryption can be employed to mitigate these issues. Although encryption is sufficient to protect digital images and videos in some civil applications, these issues are considered when advanced encryption algorithms are specifically designed for sensitive digital images and videos, because their special features differ significantly from those of texts.

Sr. No.	Paper Title	Journal Name and Year Of Application	Work Done	Technique Used	Disadvantages/Gaps	Student Name and Registration
1.	Visual Cryptography by Moni Naor and Adi Shamir	Advances in Cryptography-EUROCRYPT' 94- 2009	The easiest adaptation of the Visual Secret sharing aspects that the message comprises of an assortment of high contast pixels. Every unique pixel shows up in 'n' changed adaptations , one for every trasparency.	A new cryptographic scheme to decode concealed images without any cryptographic computations.	The advancement of security has been mentioned but hardly any informations about exception cases is found.	Anubhav Agarwal - 20BCE0714
2	A Survey on Visual Cryptography Techniques and Their Applications	International Journal Of Computer Science and Information Technologies Vol 6. (2) - 2015	This visual Cryptography method uses to images and includes both. When both the encoded images are combined, when the encoded images are combined, they form the	Based on the bluenoise detecting principles, their proposed method utilizes void and cluster algorithm to encode a	It cannot handle noisy data and outliners. It is not suitable to identify clusters with non convex shapes.	Anubhav Agarwal - 20BCE0714

			first and image, and when the combined with one of them being reversed it produces the other image.	secret binary image into halftone shares (images) carrying significant visual information	Thus resulting in reduced image quality.	
3	Secure Communication Based On Rubik's Cube Algorithm And Chaotic Baker Map	Procedia Technology - 2016	The chaotic baker map plays to role of randomizing the pixels in an image. This is done in order to improve the security of the image by using the chaotic mapping as a pre-processing layer.	The principle used here is Rubik's cube principle with chaotic Baker map is presented.	It does not have the fastest diffusion mechanism. Makes the process a bit lengthy due to having two layers.	Anubhav Agarwal - 20BCE0714
4	Rubik's cube principle based image encryption algorithm implementation	2015 - 7th International Conference on Electronics, Computers and Artificial Intelligence (ECAI)	Keys of random length are taken, and the maximum iteration for the algorithm is defined. each row and column is operated by bitwise XOR operator	Rubik's cube Algorithm Implementation	The performance depends on the image given by the user, can be unstable.	Anubhav Agarwal - 20BCE0714
5	Secure Communication Based On Rubik's Cube Algorithm	ICETEST - 2015	Two keys KR and KC of length M and N respectively, are generated randomly. Each element in KRi and KC j can take 0 to 255. We then Initialize a counter ITR which represents the number of iterations.	Chaotic Map, with R.C Algorithm	Time Consuming, other better methods available with much less complexities.	Manikonda Sri Pavani - 20BCT0283

6	DESIGN AND SIMULATION OF SECURE IMAGE ENCRYPTION ALGORITHM	IJARIIE Journal Vol-2 Issue-4 2016	I0 : α bit gray scale image. Image size is M x N. Generate random vectors KR & KC of length M & N resp. KR(i) & KC (j) elements take a random value of set A={0,1,2,3.......2α 1}.K R & KC - must not have constant value. Determine no. of iteration.	Design of encryption algorithm using R. C Algorithm	Designing the Algorithm is complex using wavelet transform s.	Manikonda Sri Pavani - 20BCT0283
7	A study on the analysis of Hill cipher in cryptogra phy	International Journal of Mathematics Trends and Technology (IJMTT) – Volume 54 Number 7- February 2018	A Complete explanation of Hill Cipher algorithm .The encryption algorithm takes m successive Plain text letters and instead substitutes m cipher letters. For encryption: C=Ek(P). For decryption: P=Dk(C). A theorem to crack Hill Cipher when the intruder captures enough plaintext along with the corresponding cipher text.	Hill Cipher Algorithm	The basic Hill Cipher can be cracked eaisly by the intruder Unfortuna tely, the basic Hill cipher is vulnerabl e to a known-plaintext attack because it is completel y linear.	Manikonda Sri Pavani - 20BCT0283
8	Image Encryption Using Advanced Hill Cipher Algorithm.	International Journal of Recent Trends in Engineering, Vol. 1, No. 1, May 2009	Image encryption using proposed advanced hill cipher technique. An involuntary key matrix of dimension m * m is constructed. The plain image is divided into m*m symmetric blocks. The ith	The principle used here is Hill cipher with the usage of involuntary matrix.	According to the results shown hill cipher alone can’t encrypt the images perfectly if they	Manikonda Sri Pavani - 20BCT0283

			pixels of each block are brought together to form a temporary block.Hill cipher technique is applied to the temporary block.		contain large area covered with same colour or gray level.	
9	Comparison of Vigenere Cipher and Affine Cipher in Three-pass Protocol for Securing Image	6th International Conference on Science and Technology (ICST), 2020	In Vigenere cipher, the keys of both parties are divided into RGB(red, green, blue) dimensions. For both ciphers, keys of range 0-100000 are randomly generated.	Image security is performed using classical cryptograph y methods. Affine cipher is used as one test model	Cryptanal ysis of all kinds are not performe d and thus, it can't be proved that this process is secure enough. No real time implemen tation.	Ritocheta Roy - 20BCI0332
10	Implementa tion of Affine Transform Method and Advanced Hill Cipher for securing digital images	2016 10th International Conference on Telecommun ic ation Systems Services and Applications (TSSA)	Affine transform is a mathematical concept used to convert coordinates of pixel of original image into new coordinates in the same image. 9 images (3 different image in 3 different format BMP, PNG, JPEG) are encrypted in this method.	Combined two methods Use affine transform first and then advanced hill transform	They used gray with 256 levels digital image. They used symmetri c key algorithm.	Ritocheta Roy - 20BCI0332
11	MULTISTAG E IMAGE ENCRYPTION USING RUBIK'S CUBE FOR SECURED IMAGE TRANSMISSI ON	Vol 7, No 4 (2019) of A. Abdullatif	Square matrix of image is taken All elements in individual rows are added if sum is even right circular shift, if sum is odd- left circular shift	Rubik's cube algorithm and steganograp hy	This is method is slightly outdated and hence better methods have been found	Ritocheta Roy - 20BCI0332

12	A Secure Image Encryption Algorithm Based on Rubik's Cube Principle	Journal of Electrical and Computer Engineerin g - 2012	The pixels of grayscale original image the principle of Rubik's cube is deployed which only changes the position of the pixels. Using two random secret keys, the bitwise XOR is applied.	Rubik's Cube Principle	This paper does not give the idea of recent developm ents in the field of image security and improvem ents in image security. These test only involve the visual testing and security analysis.	Ritocheta Roy - 20BCI0332
13	Applications and usage of visual cryptography	2016 5th International Conference on Reliability, Infocom Technologies and Optimization (Trends and Future Directions) (ICRITO)	RVCS has the capability to enhance information in images through recursive hiding	Rijndael , RC6 Block Ciphers and VCS	There are some limitation s occurring due to some informati on present in sub-pixels.	Ruksana Tabassum - 20BCE2650
14	Visual Cryptography Using Hill Cipher and Advanced Hill Cipher Techniques	Springer, Singapore 11 April 2021	It uses Involutory key matrix for encryption instead of random key matrix	Hill Cipher and Advanced Hill Cipher	It cannot encrypt images that contain large areas of a single color.	Ruksana Tabassum - 20BCE2650
15	Image Encryption: Using AES, Feature extraction and Random	IEEE Xplore - 2015	The project employs AES in two levels, LSB as feature extraction, random number	AES algorithm	Data not calculated on time and space complexit y of the	Ruksana Tabassum - 20BCE2650

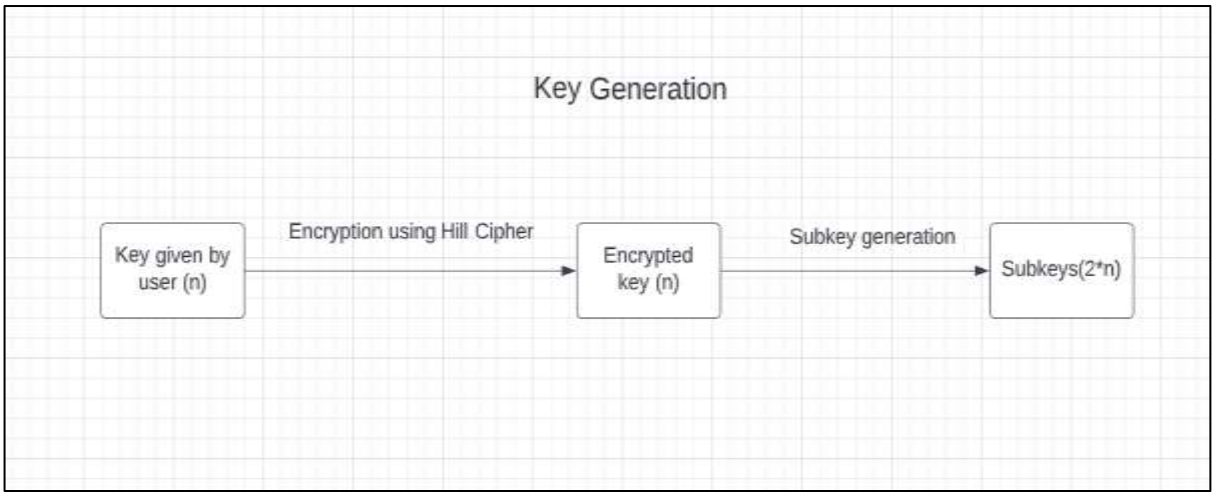
	No. generation		and key generation and encryption		algorithm used.	
16	An Improved Secure Image Encryption Algorithm Based on Rubik's Cube Principle and Digital Chaotic Cipher	Hindawi - 2013	Image encryption by chaotic pixels substitution (in order to achieve desired diffusion factor) and Rubik's cube principle based pixels permutation (in order to achieve desired confusion factor	Modified Rubik's cube algorithm	Random noise attacks affect the decrypted image.	Ruksana Tabassum - 20BCE2650

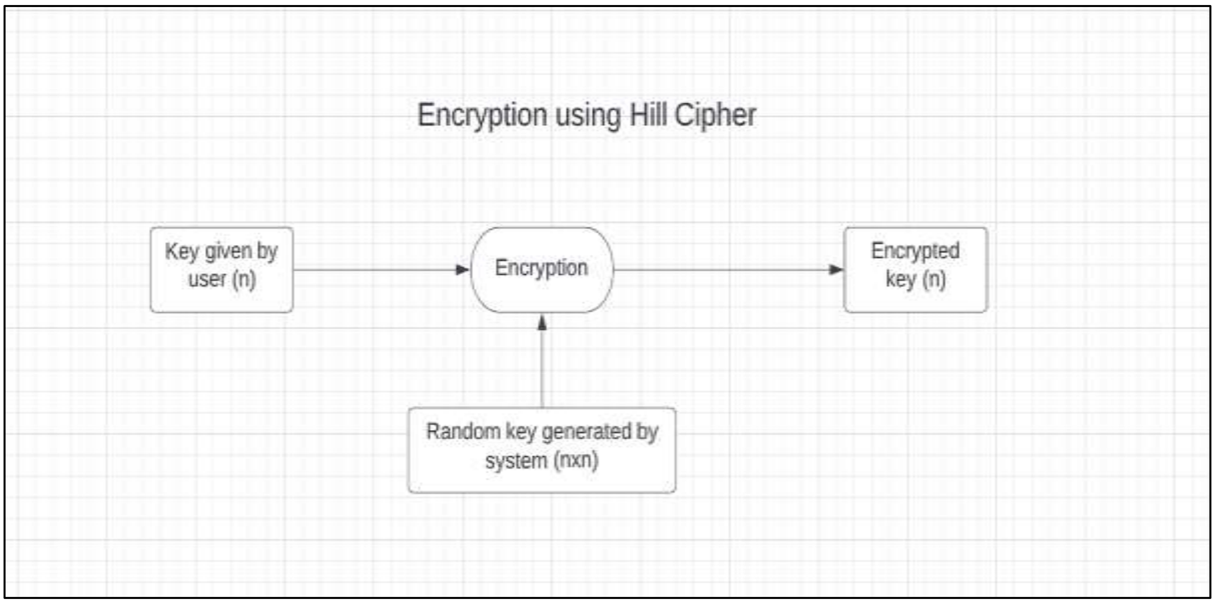
PROPOSED MODEL

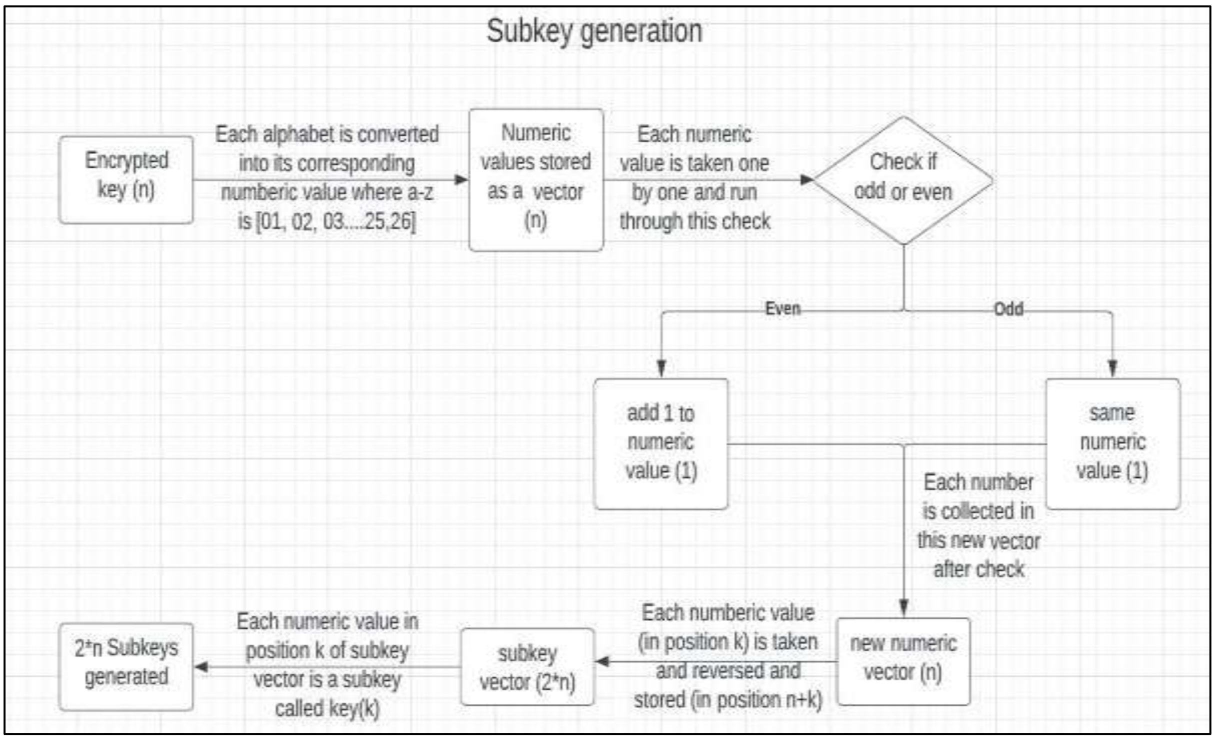
SYSTEM ARCHITECTURE

PART – 1

1. Hill cipher technique is used for the key generation process.
2. The key given by user of size n is encrypted using a key which is randomly generated of size (nxn) using Hill Cipher.
3. Now, each alphabet of encrypted key is converted into its corresponding numeric value where we are assigning the values of a-z as 01,02,03…26 respectively.
4. These numeric values(n) are stored into a vector and then each numerical value is taken individually and processed to check if it's even or odd.
5. For even: add 01 to the numerical value
6. For odd: it remains same.
7. Now, the numbers obtained in the order given are collected in the new vector and the positions of the numerical value (each single digit in the two digit number) are reversed and stored in the (n+k)th position. [Assuming, k is the position of the numerical value and n is the size of the vector].
8. This forms the subkey vector.
9. So, in total 2*n subkeys are generated which are referred as key(k). [Assuming, k is the position of the subkey.



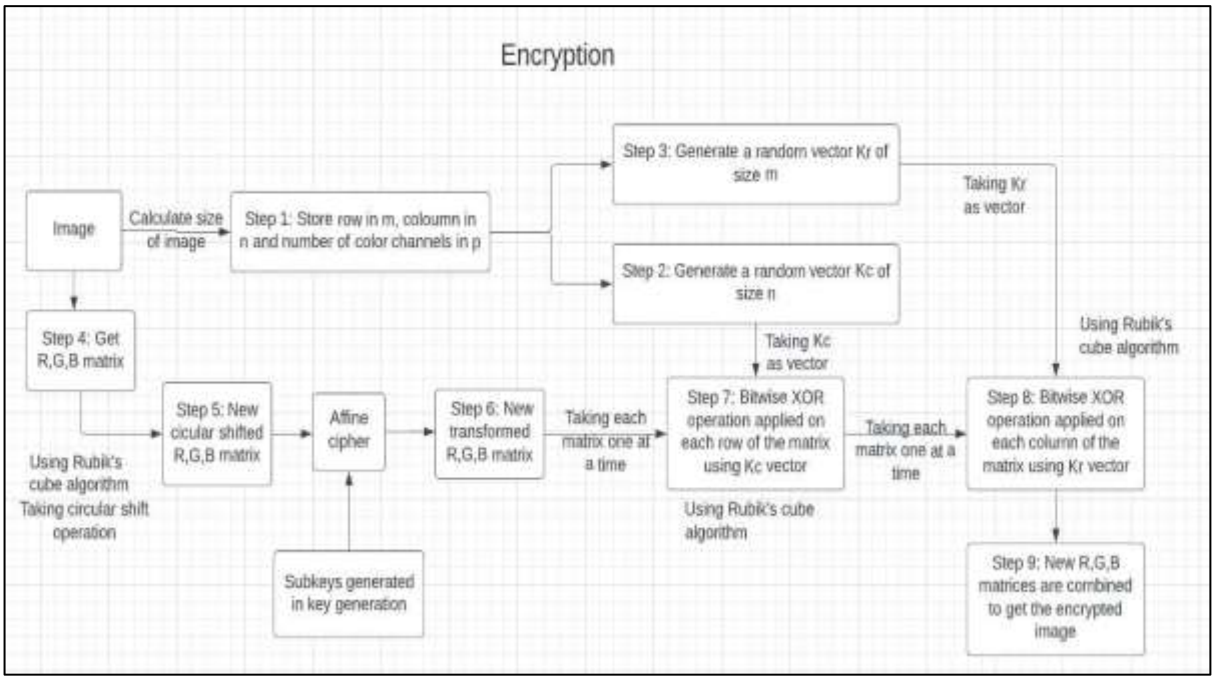

Encryption using Hill Cipher
Key given by user (n)
Encryption
Encrypted key (n)
Random key generated by system (nxn)


Subkey generation
Encrypted key (n)
Each alphabet is converted into its corresponding numberic value where a-z is [01, 02, 03....25,26]
Numeric values stored as a vector (n)
Each numeric value is taken one by one and run through this check
Check if odd or even
Even
Odd
add 1 to numeric value (1)
same numeric value (1)
Each number is collected in this new vector after check
new numeric vector (n)
Each numberic value (in position k) is taken and reversed and stored (in position n+k)
subkey vector (2*n)
Each numeric value in position k of subkey vector is a subkey called key(k)
2*n Subkeys generated

SYSTEM ARCHITECTURE

PART – 2

1. The size of image IMG is calculated and stored as m, n, and p where m, n and p are represented by rows (height), columns(width) and number of colour channels (3 for RGB) respectively.
2. Two random vectors Kr and Kc of length m, n respectively are generated.
3. The image is split into 2-D matrices to get Red, Green and Blue components (called R, G and B matrix respectively.)
4. Using rubik's cube algorithm's circular shift on each matrix, we create new scrambled R,G,B matrix. 5. Affine cipher transformation is applied on each pixel values of these new R, G and B matrices.
5. Using vector Kc, bitwise XOR operator is applied to each row of the matrices from step 5.
6. Using vector Kr, bitwise XOR operator is applied to each column of matrices from step 6.
7. Combine this final R, G and B matrices to get encrypted image EIMG.



Encryption
Image
Calculate size of image
Step 1: Store row in m, coloumn in n and number of color channels in p
Step 3: Generate a random vector Kr of size m
Step 2: Generate a random vector Kc of size n
Taking Kr as vector
Taking Kc as vector
Using Rubik's cube algorithm
Step 4: Get R,G,B matrix
Using Rubik's cube algorithm Taking circular shift operation
Step 5: New cicular shifted R,G,B matrix
Affine cipher
Subkeys generated in key generation
Step 6: New transformed R,G,B matrix
Taking each matrix one at a time
Step 7: Bitwise XOR operation applied on each row of the matrix using Kc vector
Using Rubik's cube algorithm
Taking each matrix one at a time
Step 8: Bitwise XOR operation applied on each column of the matrix using Kr vector
Step 9: New R,G,B matrices are combined to get the encrypted image

SYSTEM ARCHITECTURE

PART – 3

1.The size of encrypted image is calculated and stored as m, n, and p where m, n and p are

represented by rows (height), columns(width) and number of colour channels (3 for RGB) respectively.

2.The value of the two random vectors generated during encryption Kr and Kc of length m, n

respectively is taken as input .

3. The image is split into 2-D matrices to get Red, Green and Blue components (called R, G and

B matrix respectively.)

3. Bitwise XOR operation applied on Kr and each column of each of the matrices.

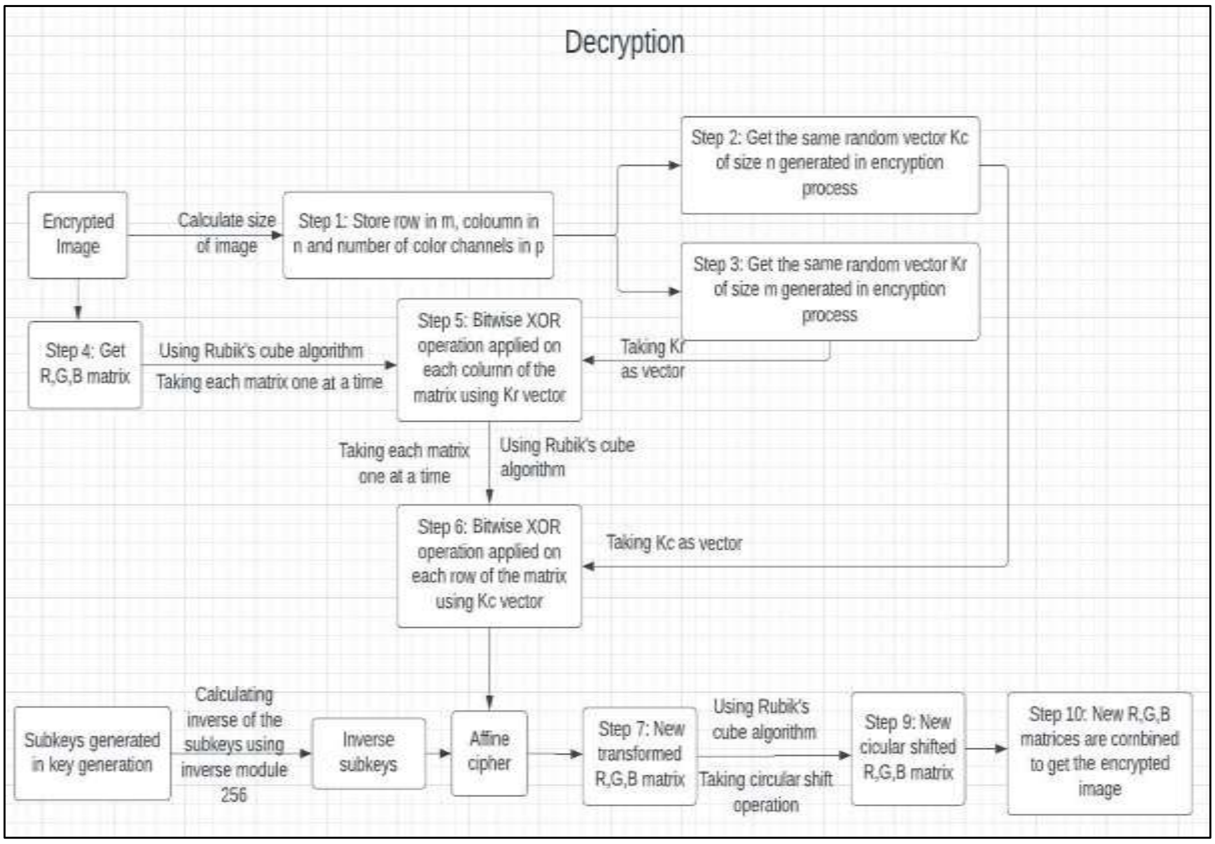
4. Bitwise XOR operation applied on Kc and each row of each of the matrices obtained from step 3.

5. Find modular inverses of the subkeys generated during key generation process using mod 256.

6. Perform affine cipher on the matrices obtained in step 4 using the inverse modulo 256 keys.

7.Perform circular shift on the matrices obtained in step 6 to get the final matrices.

8. The final matrices obtained in step 7 are combined to get the decrypted image which is same as the original encrypted image DIMG.


Decryption
Encrypted Image
Calculate size of image
Step 1: Store row in m, coloumn in n and number of color channels in p
Step 2: Get the same random vector Kc of size n generated in encryption process
Step 3: Get the same random vector Kr of size m generated in encryption process
Step 4: Get R,G,B matrix
Using Rubik's cube algorithm
Taking each matrix one at a time
Step 5: Bitwise XOR operation applied on each column of the matrix using Kr vector
Taking Kr as vector
Taking each matrix one at a time
Using Rubik's cube algorithm
Step 6: Bitwise XOR operation applied on each row of the matrix using Kc vector
Taking Kc as vector
Subkeys generated in key generation
Calculating inverse of the subkeys using inverse module 256
Inverse subkeys
Affine cipher
Step 7: New transformed R,G,B matrix
Using Rubik's cube algorithm
Taking circular shift operation
Step 9: New cicular shifted R,G,B matrix
Step 10: New R,G,B matrices are combined to get the encrypted image

LIST OF MODULES

Module 01: Manikonda Sri Pavani - 20BCT0283

Key generation

Module 02 - Anubhav Agarwal - 20BCE0714

Getting R,G,B matrix from image and scrambling it using Rubik's cube algorithm.

Module 03: Ritocheta Roy - 20BCI0332

Transforming R,G,B matrix using Affine Cipher method

Module 04 - Ruksana Tabassum - 20BCE2650

Bitwise XOR operation using Rubik's cube algorithm.

MODULE-1

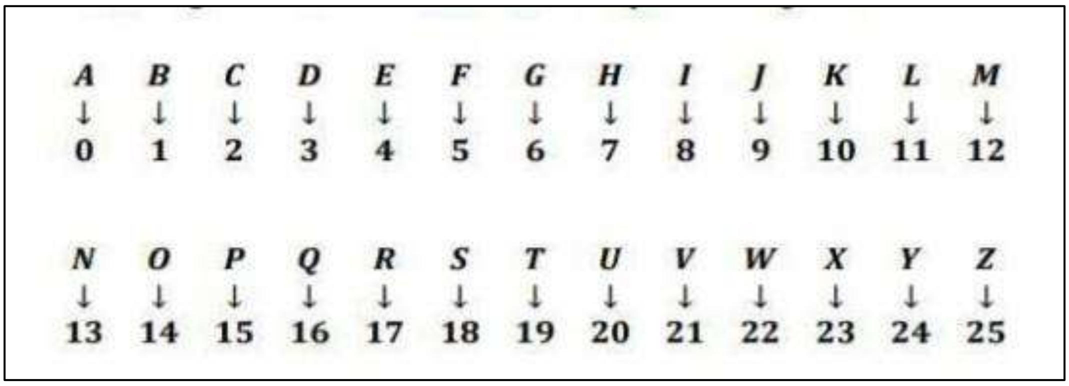
HILL CIPHER

1. The Hill cipher is based on linear algebra which is used to encrypt and decrypt the plaintext.

2.Generally ,the below mentioned structure of numbers and letters are used in hill cipher encryption,but this can be modified as per our requirement.

3. For,encrypting a message,we start with each block having 'n' letters and then multiply with (nxn) matrix,in parallel with mod 26.

4. Subsequently, for decryption, each block needs to be multiplied by the inverse matrix.



A	B	C	D	E	F	G	H	I	J	K	L	M
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
0	1	2	3	4	5	6	7	8	9	10	11	12

N	O	P	Q	R	S	T	U	V	W	X	Y	Z
↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
13	14	15	16	17	18	19	20	21	22	23	24	25

EXAMPLE

1. Given key:ACT
2. Key generated randomly for the encryption process is : GYBNQKURP(hidden key).
3. Hill Cipher is performed obtaining a Cipher Text(CT): POH.
4. Now, Each alphabet is converted into its corresponding numerical value using

LETTER	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
NUMBER	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26

5. NUMERIC VALUE OF LETTERS:16,15,08
6. Conversion New numeric values for subkey formation: 16,15,08

Subkey generation: inverse of the numbers- 61,51,80

key1 vector is: [16,15,08,61,51,80]

Six subkeys: key1(0), key1(1), … key1(5)

MODULE-2

CONSTRUCTION OF RGB MATRIX

1. The image is converted into R,G,B matrix using Matlab as a platform.
2. Taking this R,G,B matrix as input, Rubik's Cube Circular Shift Algorithm is used to scramble the matrice.
3. Circular Shift means shifting the rows from bottom rows to top rows circularly 1 times and then shifting the columns from left side to right side 10times. Similar transformation is applied to matrix R,G,B.

CONTROL SHIFT

Next we get vector k1=[key1(0) ,–key1(3)],

k2=[key1(1) –key1(4)],

k3=[key1(2), - key1(5)]

and we perform circular shift as:

i. For matrix R R(:,:)=circshift(R(:,:),k1);
ii. R(:,:)=circshift(R(:,:),[01,10]);
iii. The above code means shifting the rows from bottom rows to top rows circularly 1 times and then shifting the columns from left side to right side 10times. Similar transformation is applied to matrix G and B.

MODULE-3

AFFINE CIPHER

1. The Affine Cipher is a substitution cipher in which each letter of the English alphabet is represented by a numeric value ranging from 0 for A to 25 for Z.
2. Every alphabet is then translated to another alphabet using an encryption mechanism, which is nothing more than a mathematical formula (explained briefly).
3. During the decryption procedure, the reverse formula is used to recover the original text from the encrypted text.
4. Here, E(x) is the Affine encryption function, with x being the integer value of the English alphabet as stated above, and p and r being (appropriately chosen) numbers that work as constants and serve as the key for this Affine cipher.
5. Different variations of the Affine scheme come from changing the values of p and r. Taking mod by 26 limits the numerical value E(x) to a range between 0 and 25.
6. E(x)=(px+r)mod M.(The value of p must be chosen such that p and M are co-prime).
7. For decryption, X= p-1(y-r)mod M.

SUBSTITUTION

The general formula for the affine cipher is E(x) = (P* x + Q) mod 26 where x is any input alphabet and A and B are constants. This formula we want to use for pixel transformation so instead of 26 value used is 256 as for class type int the range of pixel values can be in between 0 and 255 i.e total 256 values. The value of x will be every pixel value from the Red ,Green and Blue matrix.

Modified Affine Cipher for Image Processing will be:

i. R(i,j)=(key1(1) * R(i,j) + key1(3)) mod 256
ii. G(i,j)=(key1(2) * G(i,j) + key1(4)) mod 256
iii. B(i,j)=(key1(3) * B(i,j) + key1(5)) mod 256

MODULE – 4

RUBIK'S CUBE XOR OPERATION

For rows: (i) If i%2==1

R(i,j)= R(i,j)^Kc(j)

Else

R(i,j)= R(i,j)^rotate180(Kc(j))

Similarly for G and B matrix

(ii) For columns:

If j%2==0

R(i,j)= R(i,j)^Kr(i)

else R(i,j)= R(i,j)^rotate180(Kr(i))

Finally at the end the 3, 2-D matrices combined to get a 2D image of size m by n by P.

CONSTRUCTION OF ENCRYPTED IMAGE

After implementing Rubik's Cube XOR operation, the final R,G,B matrices are combined to get the encrypted image.

This encrypted image will be transferred to the receiver.

The implementation of these processes will be shown in the demonstration.

RESULTS AND DISCUSSION

Encryption Code:

```
from random import randint

import numpy

from tkinter import *

from tkinter import filedialog, Text, Tk

import os

import tkinter as tk

from PIL import Image, ImageTk

from PIL import Image


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)


def encryption(im, pix):

    # 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])

    #Initial RGB Matrix Print

    #print("\nRGB Matrix: ")

    #print("\nR: ",r)

    #print("\nG: ",g)

    #print("\nB: ",b)




    # 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)]

    # Sub-key generation
    def getKeyMatrix(key, message, keyMatrix):
        k = 0
        for i in range(len(message)):
            for j in range(len(message)):
                keyMatrix[i][j] = ord(key[k]) % 65
                k += 1


    def encrypt(messageVector, message, cipherMatrix, keyMatrix):
        for i in range(len(message)):
            for j in range(1):
                cipherMatrix[i][j] = 0
                for x in range(len(message)):
                    cipherMatrix[i][j] += (keyMatrix[i][x] * messageVector[x][j])
                    cipherMatrix[i][j] = cipherMatrix[i][j] % 26

    def HillCipher(message, key):
        keyMatrix = [[0] * (len(message)) for i in range(len(message))]
```

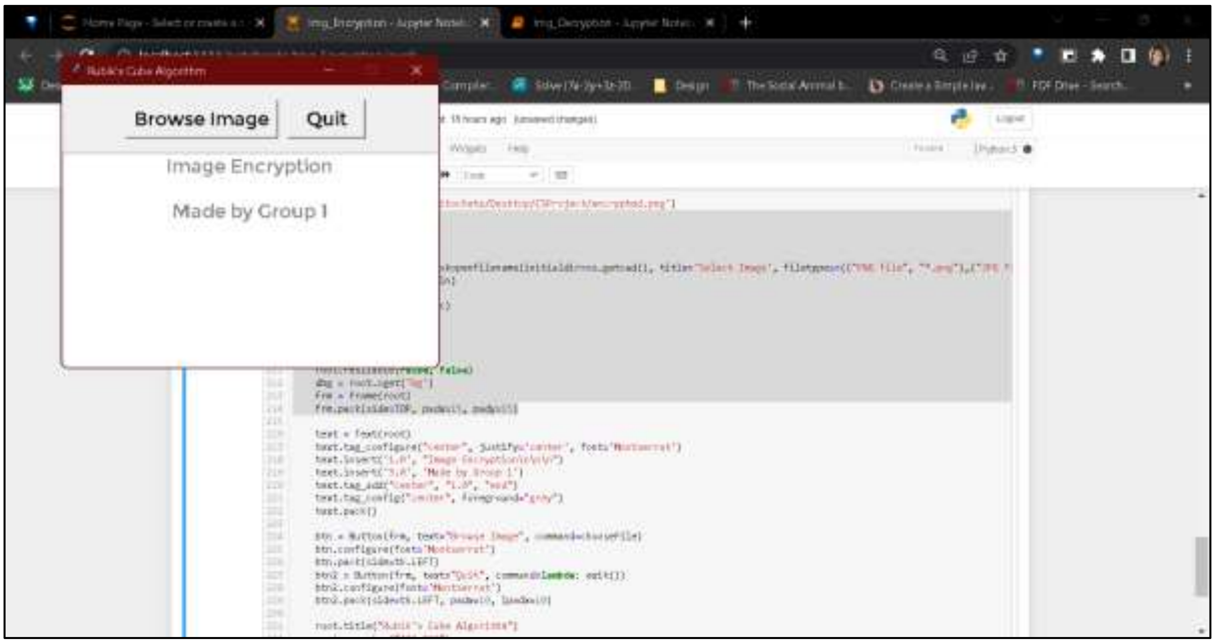
```python
    cipherMatrix = [[0] for i in range(len(message))]

    getKeyMatrix(key, message, keyMatrix)

    messageVector = [[0] for i in range(len(message))]

    for i in range(len(message)):

        messageVector[i][0] = ord(message[i]) % 65

    encrypt(messageVector, message, cipherMatrix, keyMatrix)

    CipherText = []

    for i in range(len(message)):

        CipherText.append(chr(cipherMatrix[i][0] + 65))

    temp = []

    for i in range(len(CipherText)):

        temp.append(str(ord(CipherText[i]) - 65))

    for i in range(len(temp)):

        temp.append(temp[i][::-1])

    arr = list(map(int, temp))

    return arr


message = input("Enter the private key (3 lettered):").upper()

key1 = HillCipher(message, "GYBNQKURP")

# key1=[15,14,7,51,71,7]

for i in range(3):

    if key1[i] % 2 == 0:

        key1[i] = key1[i] + 1

    if 0 <= key1[i] <= 9:

        key1[i + 3] = key1[i] * 10


# maximum number of iterations

ITER_MAX = 3
```

```
    print('\nVector Kr : \n', Kr)

    print('\nVector Kc : \n', 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):


        #Circular Shift

        # For each row

        for i in range(m):

            # right circular shift

            r[i] = numpy.roll(r[i], key1[3])

            g[i] = numpy.roll(g[i], key1[4])

            b[i] = numpy.roll(b[i], key1[5])

        # For each column

        for i in range(n):

            # up circular shift

            upshift(r, i, key1[0])

            upshift(g, i, key1[1])

            upshift(b, i, key1[2])
```

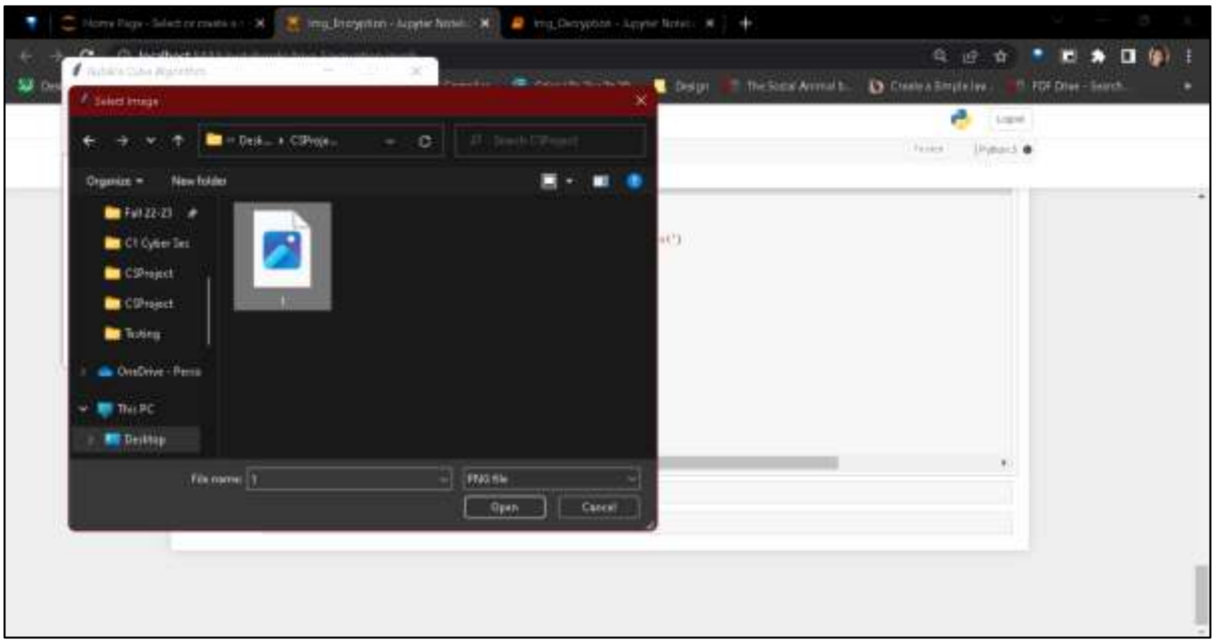
```
#RGB Matrix After Circular Shift Print
#print("\nRGB Matrix After Circular Shift: ")
#print("\nR: ",r)
#print("\nG: ",g)
#print("\nB: ",b)

# affine transformation
for i in range(m):
    for j in range(n):
        r[i][j] = (key1[0] * r[i][j] + key1[3]) % 256
        g[i][j] = (key1[1] * g[i][j] + key1[4]) % 256
        b[i][j] = (key1[2] * b[i][j] + key1[5]) % 256

#RGB Matrix After Affine Transformation Print
#print("\nRGB Matrix After Affine Transformation: ")
#print("\nR: ",r)
#print("\nG: ",g)
#print("\nB: ",b)

#XOR operation
# 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])
```

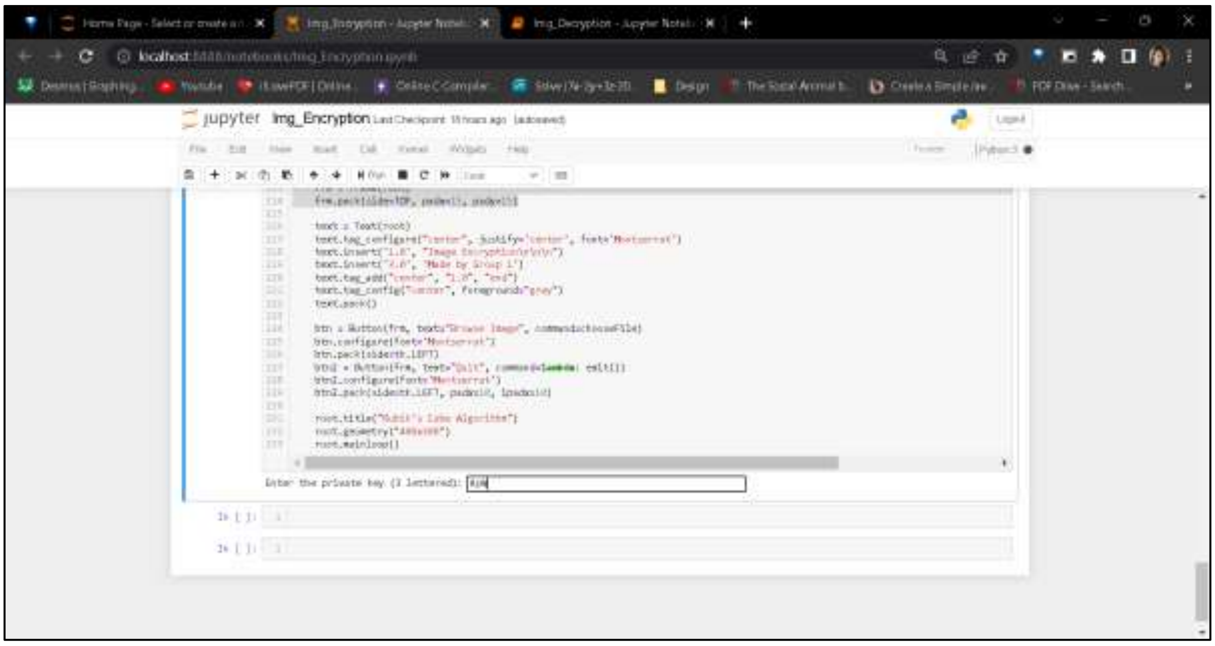
```python
                    g[i][j] = g[i][j] ^ rotate180(Kc[j])

                    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])

        #RGB Matrix After XOR operation Print

        #print("\nRGB Matrix After XOR Operation: ")

        #print("\nR: ",r)

        #print("\nG: ",g)

        #print("\nB: ",b)


        for i in range(m):

            for j in range(n):

                pix[i, j] = (r[i][j], g[i][j], b[i][j])



if __name__ == '__main__':


    def saveimg(img):

        img.save('/Users/Ritocheta/Desktop/CSProject/encrypted.png')

        print("Success")
        exit(0)


    def chooseFile():
        fln = filedialog.askopenfilename(initialdir=os.getcwd(), title='Select Image', filetypes=(("PNG file", "*.png"),("JPG File", "*.jpg"), ("All Files", "*.*")))
        img = Image.open(fln)
        pic = img.load()
        encryption(img, pic)
        saveimg(img)
        exit()


    root = Tk()
    root.resizable(False, False)
    dbg = root.cget('bg')
    frm = Frame(root)
    frm.pack(side=TOP, padx=15, pady=15)


    text = Text(root)
    text.tag_configure("center", justify='center', font='Montserrat')
    text.insert('1.0', "Image Encryption\n\n\n")
    text.insert('3.0', 'Made by Group 1')
    text.tag_add("center", "1.0", "end")
    text.tag_config("center", foreground="grey")
    text.pack()


    btn = Button(frm, text="Browse Image", command=chooseFile)
    btn.configure(font='Montserrat')
    btn.pack(side=tk.LEFT)
    btn2 = Button(frm, text="Quit", command=lambda: exit())
    btn2.configure(font='Montserrat')

    btn2.pack(side=tk.LEFT, padx=10, ipadx=10)



    root.title("Rubik's Cube Algorithm")

    root.geometry("400x300")

    root.mainloop()
```

Screenshots For Encryption:

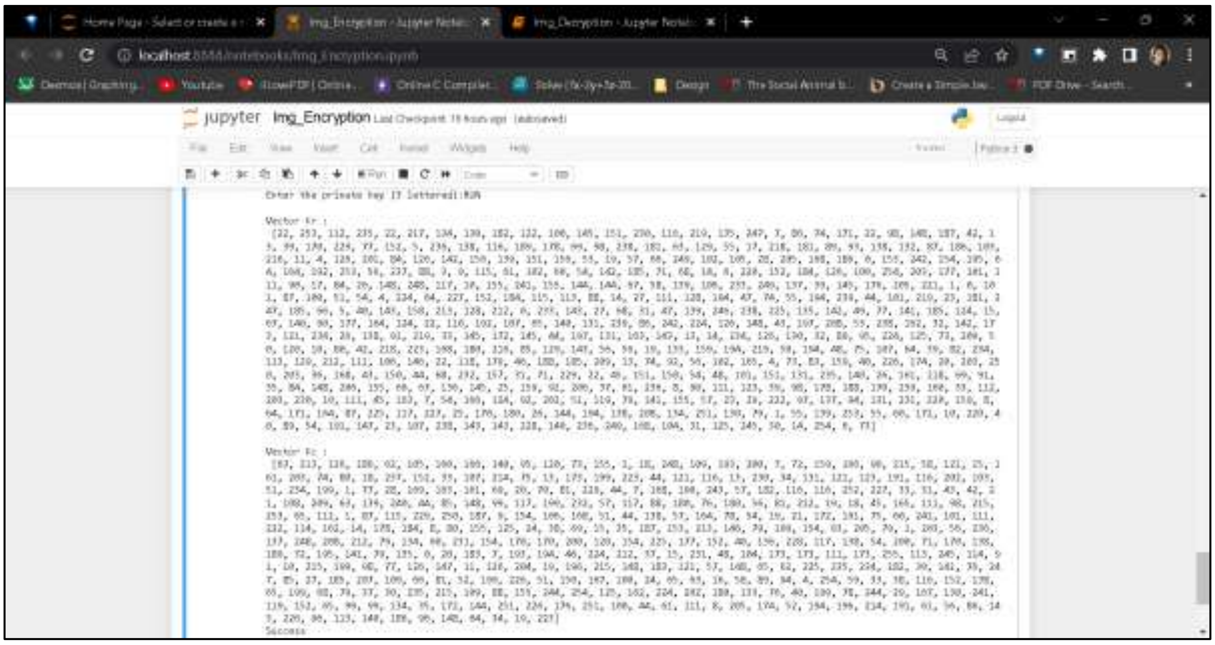


```
root.resizable(False, False)
img = root.iconbitmap('[illegible]')
frm = Frame(root)
frm.pack(side=TOP, padx=15, pady=15)

text = Text(root)
text.tag_configure("center", justify='center', font='Montserrat')
text.insert("1.0", "Image Encryption\n\n\n")
text.insert("3.0", 'Made by Group 1')
text.tag_add("center", "1.0", "end")
text.tag_config("center", foreground="grey")
text.pack()

btn = Button(frm, text="Browse Image", command=chooseFile)
btn.configure(font='Montserrat')
btn.pack(side=tk.LEFT)
btn2 = Button(frm, text="Quit", command=lambda: exit())
btn2.configure(font='Montserrat')
btn2.pack(side=tk.LEFT, padx=10, ipadx=10)

root.title("Rubik's Cube Algorithm")
```




```
frm.pack(side=TOP, padx=15, pady=15)

text = Text(root)
text.tag_configure("center", justify='center', font='Montserrat')
text.insert("1.0", "Image Encryption\n\n\n")
text.insert("2.0", "Made by Group 1")
text.tag_add("center", "1.0", "end")
text.tag_config("center", foreground="grey")
text.pack()

btn = Button(frm, text="Browse Image", command=chooseFile)
btn.configure(font='Montserrat')
btn.pack(side=tk.LEFT)
btn2 = Button(frm, text="Quit", command=lambda: exit())
btn2.configure(font='Montserrat')
btn2.pack(side=tk.LEFT, padx=10, ipadx=10)

root.title("Rubik's Cube Algorithm")
root.geometry("400x400")
root.mainloop()
```

```
Enter the private key (3 lettered): RJN
```


```
Enter the private key (3 lettered):RJN

Vector Kr :
 [illegible]

Vector Kc :
 [illegible]
Success
```

Decryption Code:

```
from PIL import Image

import numpy

from random import randint

import numpy

from tkinter import *

from tkinter import filedialog, Text, Tk

import os

import tkinter as tk

from PIL import Image, ImageTk


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:
```

```python
                a[i][j] = shiftCol[i]


def rotate180(n):
    bits = "{0:b}".format(n)
    return int(bits[::-1], 2)


def ModInv(a):
    """
        Form equation 1 = inv(a)*a mod m. we find inv(a)
        Inverse exist only if a and m 256 Coprime
    """
    for i in range(2, 256):
        if (a * i) % 256 == 1:
            return i
    return 1


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

    m = img.size[0]

    n = img.size[1]

    # print(n)

    # print(m)

    Kr = []

    Kc = []

    a = eval(input("Enter value of Kr "))

    Kr.extend(a)

    a1 = eval(input("Enter value of Kc "))

    Kc.extend(a1)

    print('Enter value of ITER_MAX')

    ITER_MAX = int(input())


    # Subkey generation

    def getKeyMatrix(key, message, keyMatrix):

        k = 0

        for i in range(len(message)):

            for j in range(len(message)):

                keyMatrix[i][j] = ord(key[k]) % 65

                k += 1


    def encrypt(messageVector, message, cipherMatrix, keyMatrix):

        for i in range(len(message)):

            for j in range(1):

                cipherMatrix[i][j] = 0

                for x in range(len(message)):

                    cipherMatrix[i][j]     +=     (keyMatrix[i][x]     *
messageVector[x][j])
```

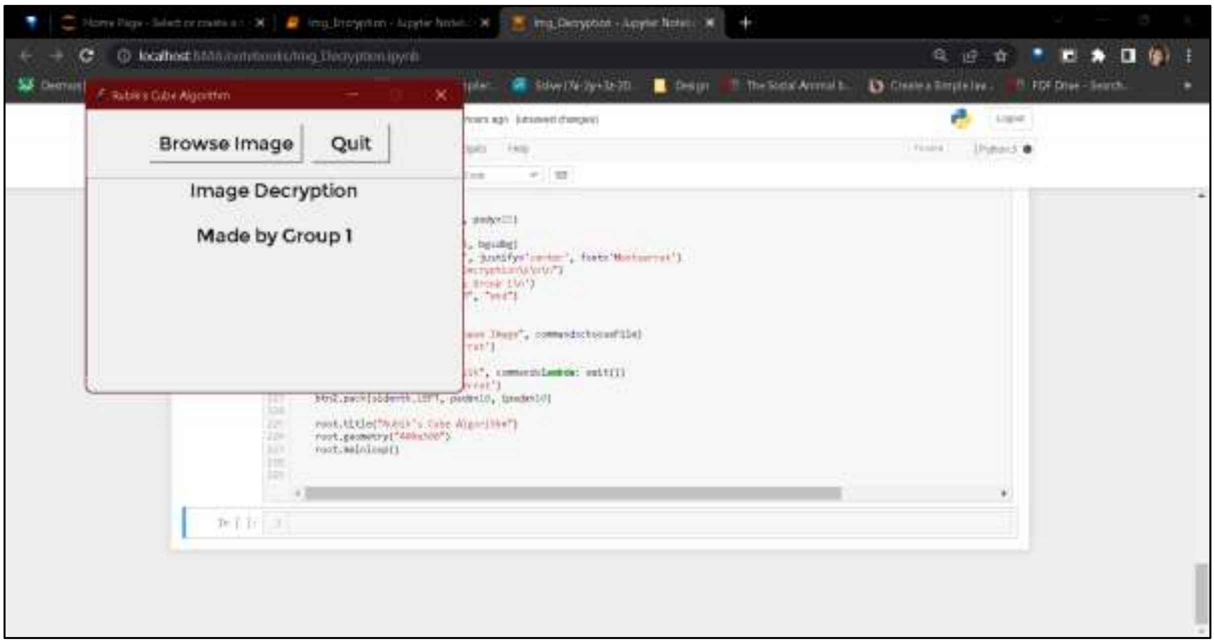
```python
            cipherMatrix[i][j] = cipherMatrix[i][j] % 26


def HillCipher(message, key):
    keyMatrix = [[0] * (len(message)) for i in range(len(message))]
    cipherMatrix = [[0] for i in range(len(message))]
    getKeyMatrix(key, message, keyMatrix)
    messageVector = [[0] for i in range(len(message))]
    for i in range(len(message)):
        messageVector[i][0] = ord(message[i]) % 65
    encrypt(messageVector, message, cipherMatrix, keyMatrix)
    CipherText = []
    for i in range(len(message)):
        CipherText.append(chr(cipherMatrix[i][0] + 65))
    temp = []
    for i in range(len(CipherText)):
        temp.append(str(ord(CipherText[i]) - 65))
    for i in range(len(temp)):
        temp.append(temp[i][::-1])
    arr = list(map(int, temp))
    return arr


message = input("Enter the private key (3 lettered):").upper()
key1 = HillCipher(message, "GYBNQKURP")
# key1=[15,14,7,51,71,7]
for i in range(3):
    if key1[i] % 2 == 0:
        key1[i] = key1[i] + 1
    if 0 <= key1[i] <= 9:
        key1[i + 3] = key1[i] * 10
```

```
inv1 = ModInv(key1[0])

inv2 = ModInv(key1[1])

inv3 = ModInv(key1[2])


for iterations in range(ITER_MAX):

    #XOR Operation

    # 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])
```

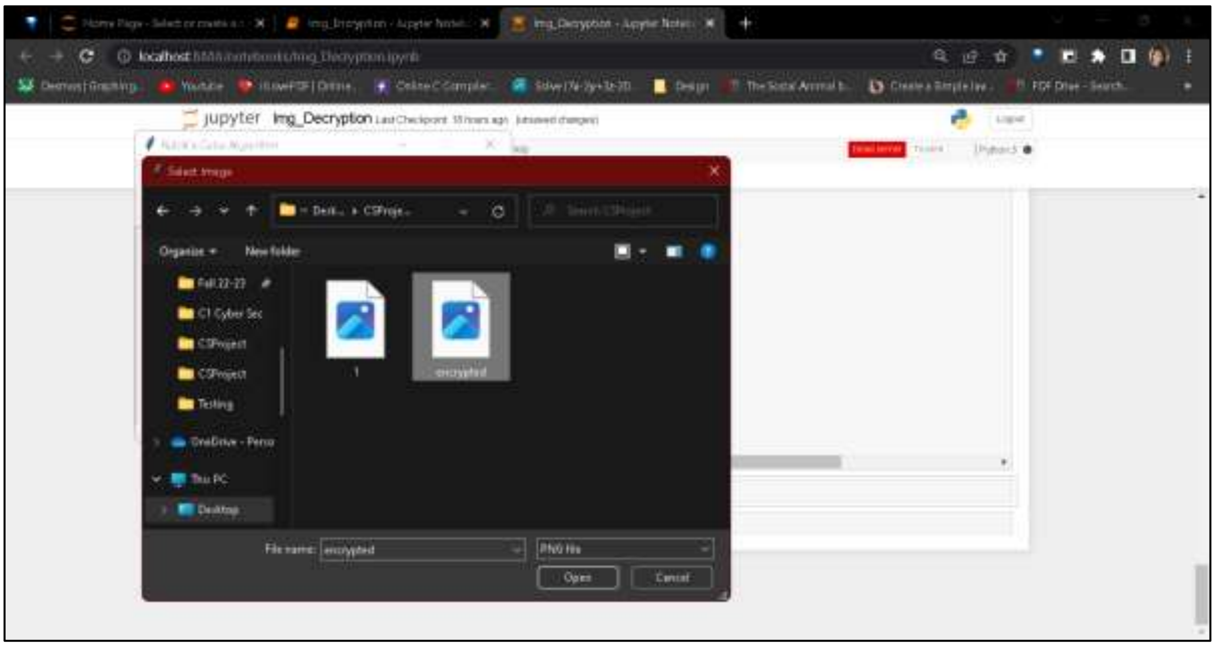
```
#Affine Transformation
for i in range(m):
    for j in range(n):
        r[i][j] = inv1 * (r[i][j] - key1[3]) % 256
        g[i][j] = inv2 * (g[i][j] - key1[4]) % 256
        b[i][j] = inv3 * (b[i][j] - key1[5]) % 256

#Circular Shift
# 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
    """
    # down circular shift
    downshift(r, i, key1[0])
    downshift(g, i, key1[1])
    downshift(b, i, key1[2])
# For each row
for i in range(m):
    """
```

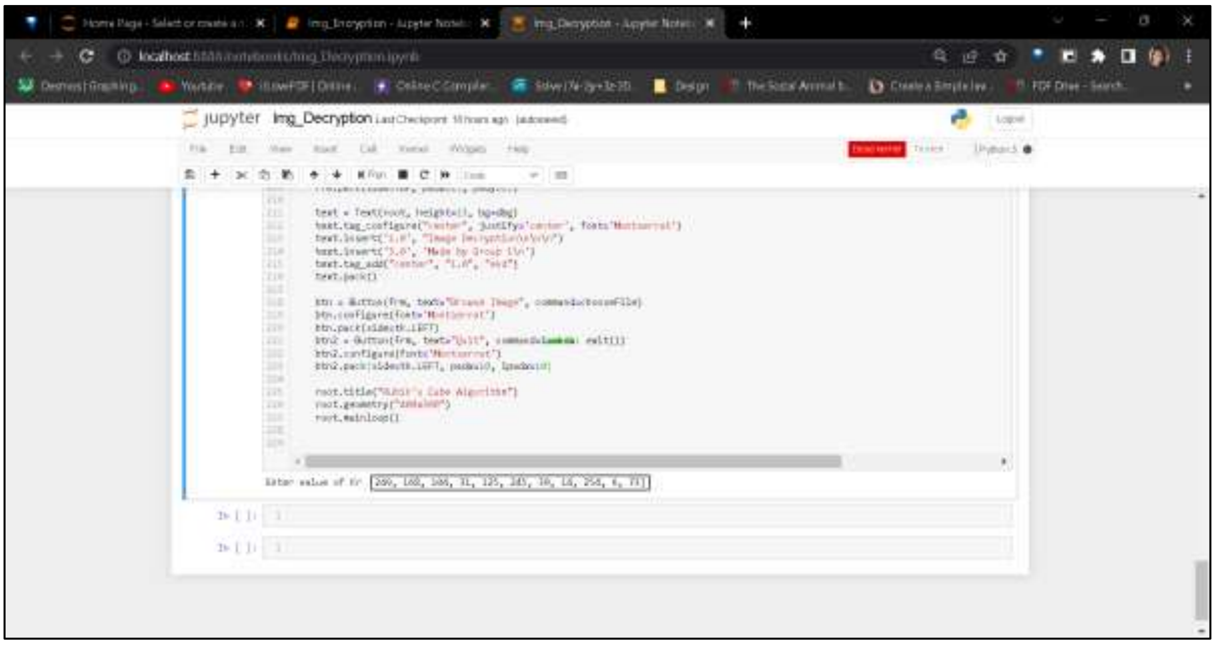
```
            rTotalSum = sum(r[i])

            gTotalSum = sum(g[i])

            bTotalSum = sum(b[i])

            rModulus = rTotalSum % 2

            gModulus = gTotalSum % 2

            bModulus = bTotalSum % 2

            """

            # left circular shift

            r[i] = numpy.roll(r[i], -key1[3])

            g[i] = numpy.roll(g[i], -key1[4])

            b[i] = numpy.roll(b[i], -key1[5])

        for i in range(m):

            for j in range(n):

                pix[i, j] = (r[i][j], g[i][j], b[i][j])



if __name__ == '__main__':

    def saveimg(img):

        img.save('/Users/Ritocheta/Desktop/CSProject/decrypted.png')

        print("Success")

        exit(0)



    def chooseFile():

        fln     =     filedialog.askopenfilename(initialdir=os.getcwd(),
title='Select  Image',filetypes=(("PNG  file",  "*.png"),("JPG  File",
"*.jpg"), ("All Files", "*.*")))

        img = Image.open(fln)

        pic = img.load()

        decryption(img, pic)
        saveimg(img)

        exit()



    root = Tk()

    root.resizable(False, False)

    dbg = root.cget('bg')

    frm = Frame(root)

    frm.pack(side=TOP, padx=15, pady=15)


    text = Text(root, height=13, bg=dbg)

    text.tag_configure("center", justify='center', font='Montserrat')

    text.insert('1.0', "Image Decryption\n\n\n")

    text.insert('3.0', 'Made by Group 1\n')

    text.tag_add("center", "1.0", "end")

    text.pack()


    btn = Button(frm, text="Browse Image", command=chooseFile)

    btn.configure(font='Montserrat')

    btn.pack(side=tk.LEFT)

    btn2 = Button(frm, text="Quit", command=lambda: exit())

    btn2.configure(font='Montserrat')

    btn2.pack(side=tk.LEFT, padx=10, ipadx=10)


    root.title("Rubik's Cube Algorithm")

    root.geometry("400x300")

    root.mainloop()
```

Screenshots For Decryption:



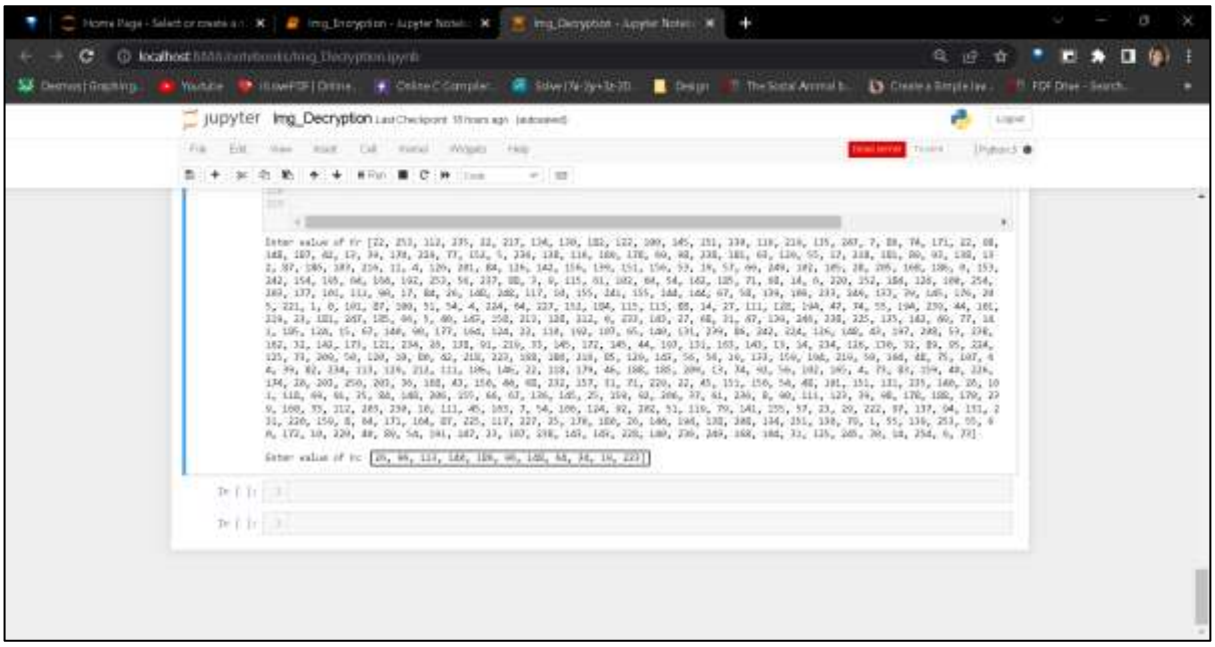
```
btn2.pack(side=tk.LEFT, padx=10, ipadx=10)

root.title("Rubik's Cube Algorithm")
root.geometry("400x500")
root.mainloop()
```





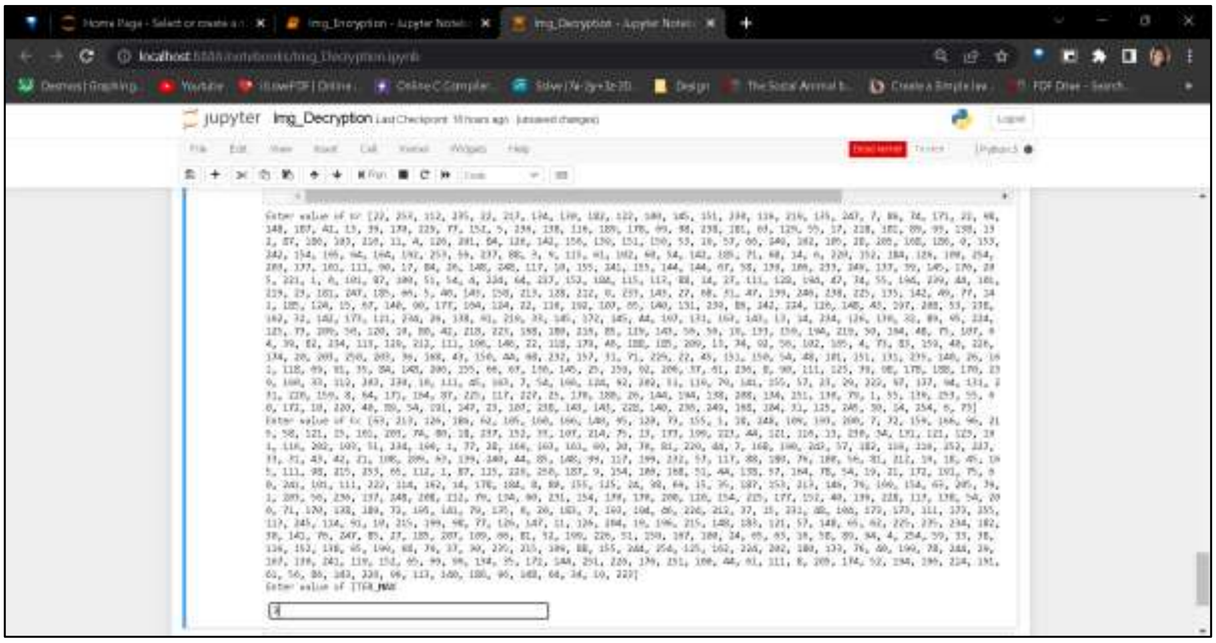
```
text = Text(root, height=11, bg=dbg)
text.tag_configure("center", justify='center', font='Montserrat')
text.insert('1.0', "Image Decryption\n\n\n")
text.insert('5.0', 'Made by Group 11\n')
text.tag_add("center", "1.0", "end")
text.pack()

btn = Button(frm, text="Browse Image", command=choseFile)
btn.configure(font='Montserrat')
btn.pack(side=tk.LEFT)
btn2 = Button(frm, text="Quit", command=lambda: exit())
btn2.configure(font='Montserrat')
btn2.pack(side=tk.LEFT, padx=10, ipadx=10)

root.title("Rubik's Cube Algorithm")
root.geometry("600x500")
root.mainloop()
```

```
Enter value of Kr [200, 168, 166, 31, 125, 245, 30, 16, 254, 6, 73]
```

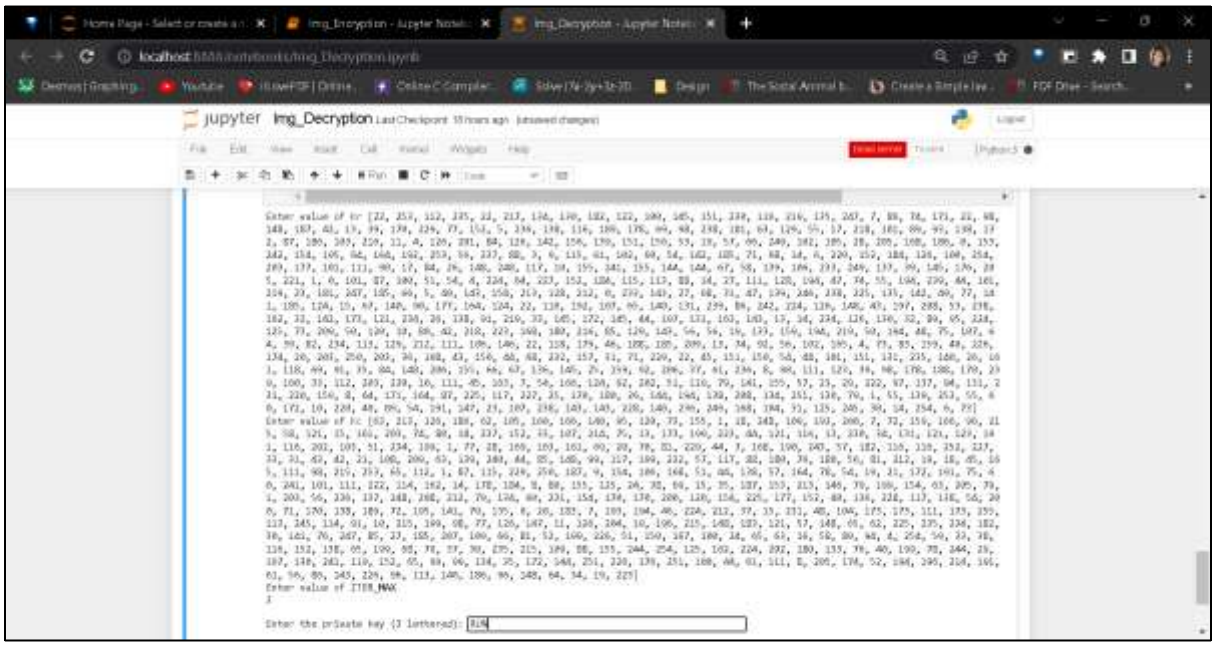


```
Enter value of Kr [22, 253, 112, 235, 32, 217, 134, 130, 182, 122, 109, 145, 151, 230, 118, 218, 135, 207, 7, 88, 74, 171, 22, 88,
148, 187, 42, 17, 34, 178, 224, 77, 152, 5, 234, 138, 116, 180, 178, 69, 98, 238, 181, 63, 120, 55, 17, 218, 181, 80, 97, 138, 13
2, 87, 185, 187, 214, 11, 4, 120, 201, 84, 126, 142, 156, 139, 151, 156, 53, 19, 57, 69, 249, 182, 105, 28, 205, 166, 186, 9, 153,
242, 154, 105, 66, 158, 192, 253, 56, 237, 88, 3, 9, 115, 61, 182, 64, 54, 162, 125, 71, 68, 14, 6, 220, 252, 184, 128, 180, 254,
189, 137, 161, 111, 90, 17, 84, 26, 140, 248, 117, 58, 155, 241, 155, 144, 166, 67, 58, 139, 188, 233, 146, 133, 79, 145, 176, 28
5, 221, 1, 0, 101, 87, 300, 51, 54, 4, 224, 64, 227, 152, 194, 115, 113, 88, 14, 27, 111, 128, 194, 47, 34, 55, 194, 239, 44, 181,
224, 23, 181, 247, 185, 46, 5, 40, 147, 258, 213, 128, 112, 6, 233, 143, 27, 68, 31, 47, 139, 246, 238, 225, 135, 143, 69, 77, 18
1, 105, 128, 15, 67, 148, 90, 177, 164, 124, 22, 118, 192, 107, 65, 140, 131, 239, 86, 242, 224, 126, 148, 42, 147, 288, 59, 238,
182, 32, 143, 173, 121, 234, 28, 138, 91, 219, 33, 145, 172, 145, 44, 193, 131, 163, 143, 13, 14, 234, 128, 136, 32, 89, 95, 224,
125, 73, 209, 58, 120, 39, 86, 42, 218, 223, 188, 188, 218, 85, 120, 147, 56, 58, 16, 133, 150, 198, 219, 59, 168, 48, 75, 187, 4
4, 39, 82, 234, 113, 119, 212, 111, 186, 146, 22, 118, 179, 46, 188, 185, 209, 13, 74, 48, 56, 182, 165, 4, 73, 83, 159, 48, 226,
134, 28, 203, 250, 203, 36, 188, 43, 156, 48, 68, 232, 157, 11, 71, 220, 22, 45, 151, 156, 58, 48, 181, 151, 131, 235, 140, 26, 10
1, 118, 69, 61, 25, 88, 140, 206, 155, 66, 67, 126, 145, 25, 159, 68, 206, 37, 61, 236, 8, 90, 111, 123, 39, 48, 170, 182, 170, 23
9, 168, 35, 212, 285, 230, 16, 111, 45, 183, 7, 54, 186, 124, 82, 282, 51, 119, 79, 141, 235, 57, 23, 29, 222, 87, 137, 94, 131, 2
31, 226, 159, 8, 64, 171, 164, 87, 225, 117, 227, 25, 178, 188, 26, 146, 196, 138, 188, 134, 251, 138, 70, 1, 55, 136, 253, 55, 6
0, 172, 18, 229, 48, 80, 54, 141, 147, 23, 187, 238, 147, 149, 228, 140, 236, 249, 168, 184, 31, 125, 245, 30, 18, 254, 6, 73]
```

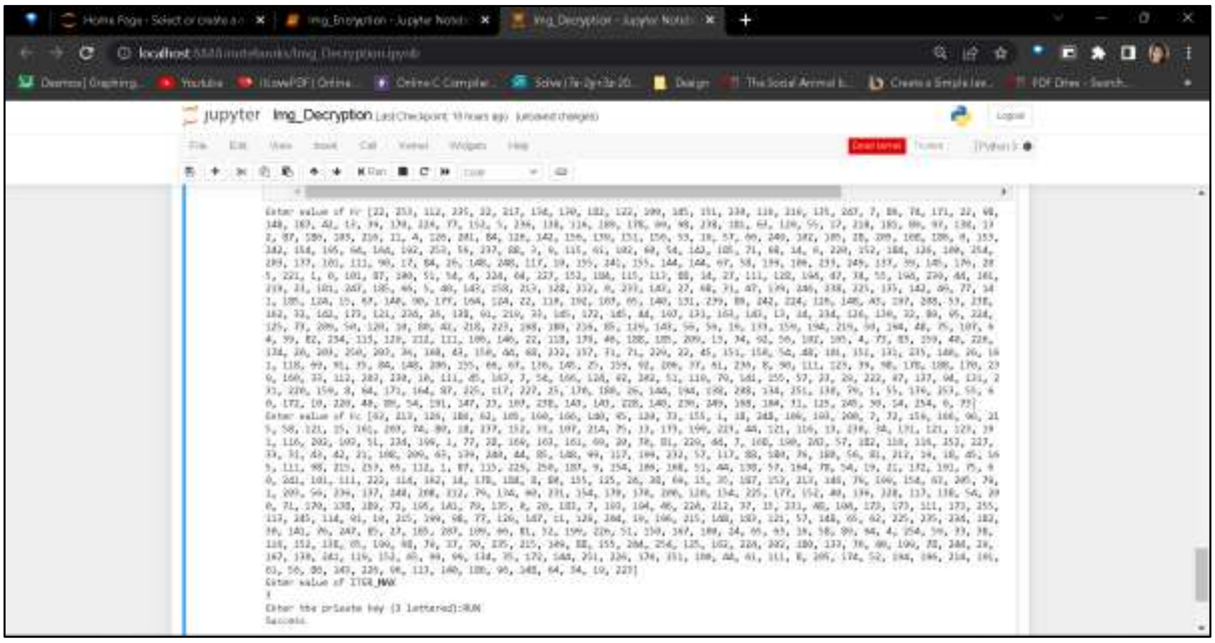
```
Enter value of Kc [26, 66, 113, 148, 106, 96, 148, 64, 34, 14, 223]
```



```
Enter value of tr [illegible]
Enter value of tc [illegible]
Enter value of ITER_MAX
3
```



```
Enter value of tr [illegible]
Enter value of tc [illegible]
Enter value of ITER_MAX
3

Enter the private key (3 lettered): RUN
```



```
Enter value of Kr [22, 253, 112, 235, 22, 217, 158, 130, 182, 122, 109, 185, 151, 230, 118, 216, 135, 207, 7, 88, 78, 171, 22, 98,
148, 187, 42, 17, 39, 170, 224, 77, 152, 5, 236, 138, 116, 189, 178, 99, 98, 238, 181, 63, 120, 55, 17, 218, 185, 80, 97, 138, 13
2, 87, 180, 185, 210, 11, 4, 126, 201, 84, 128, 142, 156, 139, 151, 156, 53, 18, 57, 60, 240, 182, 185, 28, 209, 168, 186, 8, 153,
182, 158, 165, 66, 168, 192, 253, 56, 237, 88, 3, 6, 115, 61, 182, 60, 54, 142, 185, 71, 68, 14, 6, 220, 152, 184, 126, 109, 254,
209, 137, 181, 111, 90, 17, 84, 26, 148, 208, 117, 19, 155, 241, 155, 144, 144, 67, 58, 199, 186, 233, 149, 137, 99, 145, 176, 28
5, 221, 1, 0, 101, 87, 180, 51, 54, 4, 224, 64, 227, 153, 188, 115, 117, 88, 18, 27, 111, 128, 196, 47, 38, 55, 198, 239, 44, 181,
219, 21, 101, 247, 185, 46, 5, 48, 143, 158, 213, 128, 232, 8, 233, 143, 27, 68, 31, 47, 139, 246, 238, 225, 135, 142, 89, 77, 18
1, 185, 128, 15, 67, 148, 90, 177, 164, 124, 22, 118, 192, 187, 65, 140, 131, 239, 89, 242, 224, 128, 148, 45, 197, 288, 53, 238,
182, 33, 162, 173, 121, 236, 26, 138, 61, 210, 39, 165, 172, 145, 44, 167, 131, 163, 143, 13, 14, 234, 126, 130, 32, 89, 95, 224,
125, 73, 209, 50, 128, 38, 80, 42, 218, 223, 198, 180, 216, 85, 129, 148, 56, 59, 16, 133, 159, 194, 219, 59, 144, 48, 75, 107, 6
4, 99, 82, 234, 115, 128, 212, 111, 106, 146, 22, 118, 179, 40, 188, 185, 209, 13, 34, 92, 56, 182, 185, 4, 73, 83, 159, 40, 228,
134, 26, 203, 250, 207, 36, 168, 43, 150, 66, 68, 232, 137, 71, 71, 229, 22, 45, 151, 158, 56, 48, 181, 152, 131, 235, 146, 26, 16
1, 118, 49, 91, 35, 84, 148, 209, 155, 68, 67, 136, 145, 25, 159, 92, 196, 37, 61, 239, 8, 90, 111, 123, 39, 98, 178, 188, 170, 23
0, 160, 37, 112, 283, 238, 18, 111, 45, 182, 7, 56, 166, 128, 62, 182, 51, 110, 79, 181, 255, 57, 23, 29, 222, 87, 137, 94, 131, 2
31, 220, 150, 8, 64, 171, 164, 97, 225, 117, 227, 25, 176, 188, 26, 148, 194, 198, 208, 134, 251, 138, 79, 1, 55, 176, 253, 55, 6
0, 172, 18, 220, 48, 89, 54, 191, 147, 23, 187, 238, 143, 143, 228, 148, 239, 249, 198, 184, 31, 125, 245, 38, 14, 234, 6, 75]
Enter value of Kc [82, 213, 126, 188, 62, 185, 160, 166, 140, 45, 120, 33, 155, 1, 18, 248, 186, 183, 200, 7, 72, 156, 166, 90, 21
5, 58, 121, 25, 161, 203, 74, 80, 18, 137, 152, 39, 107, 214, 75, 13, 173, 199, 223, 44, 121, 116, 13, 238, 34, 131, 121, 129, 19
1, 116, 202, 109, 51, 234, 196, 1, 77, 28, 169, 163, 161, 69, 39, 78, 81, 229, 44, 7, 168, 190, 243, 57, 182, 118, 116, 152, 227,
33, 31, 43, 42, 21, 188, 209, 63, 139, 244, 44, 85, 148, 49, 117, 194, 232, 57, 117, 88, 180, 76, 189, 56, 81, 212, 14, 18, 45, 16
5, 111, 98, 215, 257, 65, 112, 1, 87, 115, 225, 250, 187, 9, 154, 186, 198, 51, 44, 138, 57, 184, 78, 54, 19, 21, 172, 191, 75, 9
0, 241, 101, 111, 222, 114, 182, 14, 178, 188, 8, 88, 155, 125, 26, 38, 64, 15, 35, 187, 152, 213, 148, 76, 100, 154, 67, 245, 74,
1, 209, 56, 234, 137, 248, 208, 112, 79, 134, 40, 231, 154, 178, 178, 206, 128, 134, 225, 177, 152, 40, 136, 328, 113, 138, 56, 29
0, 71, 170, 138, 189, 72, 195, 141, 79, 135, 8, 20, 183, 7, 193, 194, 46, 226, 212, 37, 11, 231, 46, 166, 173, 173, 111, 173, 255,
113, 245, 114, 91, 19, 215, 199, 98, 77, 126, 147, 11, 128, 204, 19, 196, 215, 148, 183, 121, 57, 148, 65, 62, 225, 235, 234, 182,
38, 141, 76, 247, 85, 27, 185, 287, 109, 49, 81, 52, 199, 226, 51, 150, 167, 188, 24, 65, 63, 18, 58, 89, 94, 4, 254, 59, 33, 78,
118, 152, 138, 65, 190, 98, 78, 37, 70, 235, 215, 149, 88, 155, 204, 254, 125, 162, 228, 202, 180, 137, 78, 40, 199, 78, 248, 28,
167, 138, 241, 119, 152, 65, 99, 96, 138, 35, 172, 144, 251, 226, 174, 151, 108, 44, 61, 111, 8, 205, 174, 52, 198, 196, 218, 191,
63, 56, 80, 147, 226, 98, 113, 140, 188, 95, 148, 64, 34, 19, 227]
Enter value of ITER_MAX
3
Enter the private key (3 lettered):RUN
Success
```

```
```

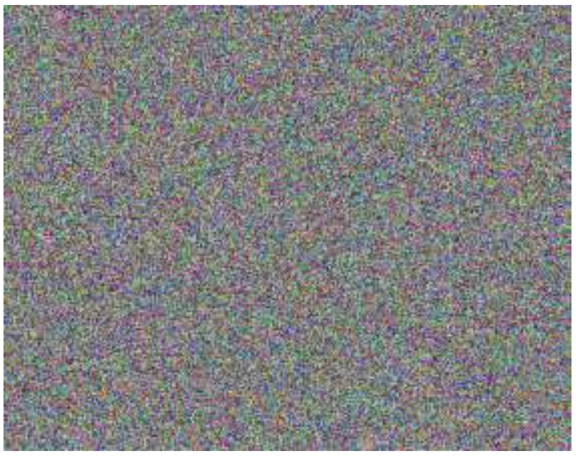
SAMPLE:

Example – 1:

Original Image

Image after decryption

Image after encryption

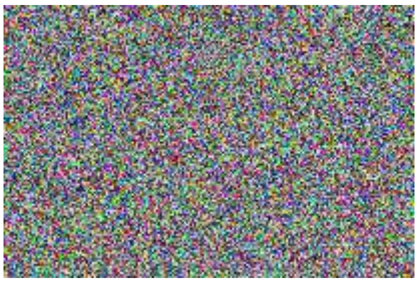


<u>Example – 2:</u>

Original Image

Image after decryption

Image after encryption



<u>Example – 3:</u>

Original Image

Image after decryption

Image after encryption

COMPARISON AND ANALYSIS

Sl. No.	Image Name	Original Image Size (Kb)	Encrypted Image Size (Kb)	%Increase in Size (%)
1	1.png	75	544	625.33
2	2.png	66	96	45.45
3	3.png	4	149	3625

Table1: Comparision of Original and Encrypted Image

Sl. No.	Image Name	Original Image Size (Kb)	Decrypted Image Size (Kb)	%Increase/Decrease in Size
1	1.png	75	80	6.66
2	2.png	66	59	-10.60
3	3.png	4	12	200

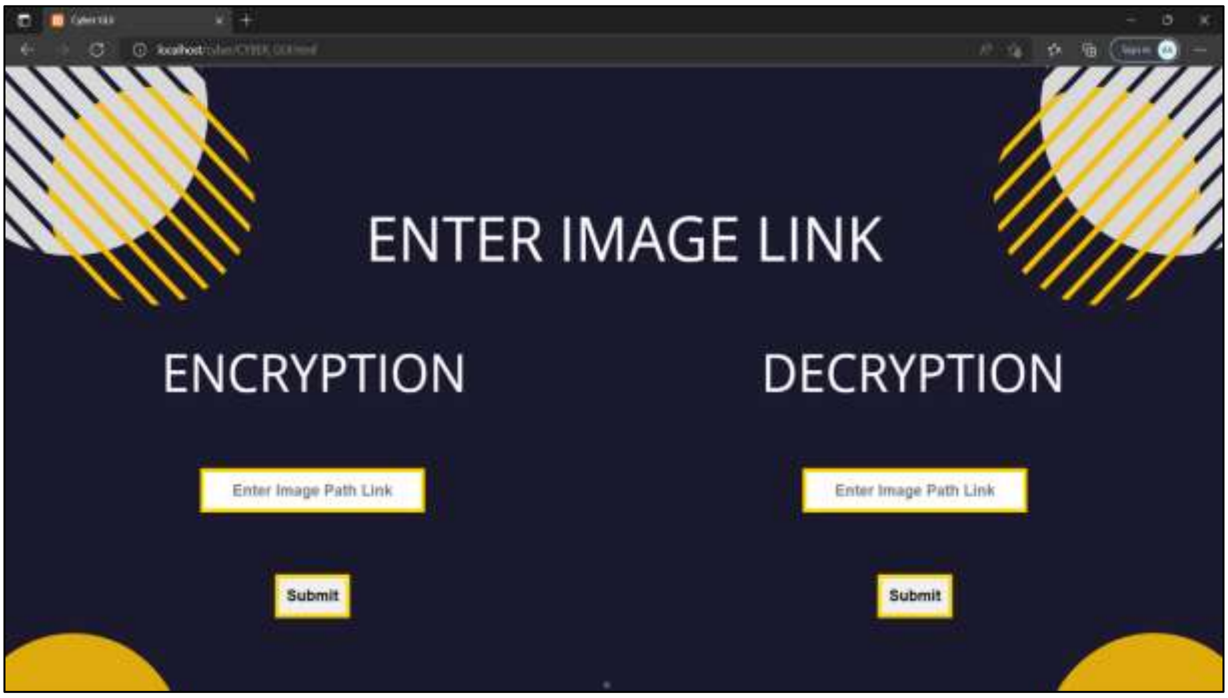
Table 2: Comparision of Original and Decrypted Image

CONCLUSION

- After observation of the results, we realized that the size of the encrypted image is more than the original image.
- As far as the comparision of the size of encrypted image and the decrypted image is seen, the decrypted image is lesser than the encrypted image.
- Since the encryption and decryption that we have compiled is a combination of several different encryption and decryption processes and methods to achieve the highest level of security and safety, the process is becoming very slow and the background manipulations and calculations are creating a heavy load of space and time complexity on the compilier and its system. Due to this load – large size images are taking almost infinite time as the matrix compilations of an image of large size make it very slow to produce fast results. Therefore we are working with small sized images for the meantime.
- Although as far as a benefit is concerned, the size of the encrypted image is more than the original image, so our system is also increasing its security agenda by making the file bulkier thus more difficult to be disrupted.
- For the overall process of encryption and decryption, the system idea is working without errors and loopholes, we achieved the goal of increased security protection during the process of encryption and decryption successfully.

FUTURE WORKS

- To overcome the boundations on the image size we have somehow by either altering the image or adding an image compressor somehow when the user drops the image to work on and encrypt it.
- Another improvement to focus on can be that the decrypted image also does not have a increased size, either have the same size as original image, or better, have a reduced size in all scenarios.
- Improve and somehow connect a better GUI that is a website portal made using HTML and CSS and configure our system into that portal, this would make the system easily accessible for people as it can be launched in form of a website and not just that but the user interface can then also be much more friendly than before as it can be beautified at a much higher scale using CSS. For reference, we did try to do that and make the portal for the same too, but the configuration of our system on that portal wasn't supported as it needs to be in any language that works on the functionalities of the website like JavaScript. Although we did manage to take the first step and make an improved GUI for the user, which shall be implemented in the future.



GUI Code:

```
<!DOCTYPE html>
<html>

<head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="description" content="">
    <meta name="viweport" content="width=device-width, initial-scale=1">
    <style>
        body{
            background-image: url("A.png");
            background-repeat: no-repeat;
            background-attachment: fixed;
            background-size: cover;
        }
        .flex-container{
            display: flex;
            text-align: center;
        }

        .flex-child{
            flex: 1;
            text-align: center;
            padding: 5%;
        }

        .link-box{
            margin-top: 65%;
            position: relative;
        }

        .button-box{
            position: relative;
        }

        input{
            border: 5px solid gold;
            padding: 10px;
            font-size: larger;
            text-align: center;
            font-weight: bolder;
        }
    </style>
    <title>Cyber GUI</title>
</head>

<body>
```

```
    <div>
        <div class="flex-container">
            <div class="flex-child enc">
                <form action="cyber.php" method="post">
                        <div class="link-box">
                            <input type="text" name="text1" placeholder="Enter
Image Path Link">
                        </div>
                        <div class="button-box">
                            <br><br><br><br><input type="submit" name =
"submit" class="login-btn" value="Submit">
                        </div>
                </form>
            </div>
            <div class="flex-child dec">
                <form action="cyber.php" method="post">
                    <div class="link-box">
                        <input type="text" name="text2" placeholder="Enter
Image Path Link">
                    </div>
                    <div class="button-box">
                        <br><br><br><br><input type="submit" name = "submit"
class="login-btn" value="Submit">
                    </div>
                </form>
            </div>
        </div>
    </div>
</body>

</html>
```

Backend Code For Image Data Storing:

```
<?php
$server="localhost";
$username="root";
$password="root";
$dbname="cyber_db";

$conn=mysqli_connect($server, $username, $password, $dbname);

if(isset($_POST['submit']))
{
    if(!empty($_POST['text1']))
    {
        $text1 = $_POST['text1'];
        $query1 = "insert into enc(link) values('$text1')";

        $run1 = mysqli_query($conn,$query1) or die(mysqli_error($conn));

        if($run1)
        {
            echo "Entry Added";
        }
        else
        {
            echo "Entry Not Added";
        }

    }
    elseif(!empty($_POST['text2']))
    {
        $text2 = $_POST['text2'];

        $query2 = "insert into decr(link) values('$text2')";

        $run2 = mysqli_query($conn,$query2) or die(mysqli_error($conn));

        if($run2)
        {
            echo "Entry Added";
        }
        else
        {
            echo "Entry Not Added";
        }

    }
    else
    {
        echo "Input Required";

    }
}
?>
```

REFERENCES

- https://www.hindawi.com/journals/mpe/2013/848392/
- https://ieeexplore.ieee.org/document/7359286
- https://ieeexplore.ieee.org/abstract/document/7301247
- https://ieeexplore.ieee.org/document/9732873
- http://www.ijmttjournal.org/2018/Volume-54/number-7/IJMTT-V54P562.pdf
- https://www.researchgate.net/publication/305109851_Secure_Communication_Based_on_Rubik's_Cube_Algorithm_and_Chaotic_Baker_Map
- https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.379.2956&rep=rep1&type=pdf
- https://ieeexplore.ieee.org.egateway.vit.ac.in/document/9732873?arnumber=9732873
- https://ieeexplore.ieee.org.egateway.vit.ac.in/document/7871068

Adrian-Viorel Diaconu and Khaled Loukhaoukha, (13 Mar 2013). An Improved Secure Image Encryption Algorithm Based on Rubik's Cube Principle and Digital Chaotic Cipher .Hindawi

- https://www.hindawi.com/journals/mpe/2013/848392/

Akanksha Upadhyaya; Vinod Shokeen; Garima Srivastava,(2015). Image encryption: Using AES, feature extraction and random no. generation. IEEE

- https://ieeexplore.ieee.org/document/7359286

N.Vijayaraghavan, S.Narasimhan, M.Baskar ,(2018).A Study on the Analysis of Hill's Cipher in Cryptography. International Journal of Mathematics Trends and Technology .

- http://www.ijmttjournal.org/2018/Volume-54/number-7/IJMTT-V54P562.pdf

Rahma Isnaini Masya; Rizal Fathoni Aji; Setiadi Yazid (2020).Comparison of Vigenere Cipher and Affine Cipher in Three-pass Protocol for Securing Image. IEEE.

- https://ieeexplore.ieee.org/document/9732873