



Acharya Prafulla Chandra College

New Barrackpore, Kolkata – 700131, West Bengal
Affiliated to West Bengal State University

Project Report

ENCRYPTION AND DECRYPTION OF AN IMAGE

.....

*Submitted for partial Fulfillment of the requirement for the degree of
Bachelor of Science in Computer Science*

*Under the supervision of
Prof. Soumya Pratik Saha*

Submitted by

1. DEEPANJAN BISWAS .

Reg. No.: 1011911401044

Roll No.: 6221101-00981

2. ISHA SHARMA.

Reg. No.: 1011921401055

Roll No.: 6222101-0098

INDEX

Sl no.	Topic	Page No.
1	Acknowledgement	02
2	Certificate	03
3	Abstract	04
4	Introduction	05
5	Logistic Map	06
6	DNA Encryption	07
7	How Logistic map and DNA works	08
8	Tools Used	09
9	Image Encryption Algorithm	09
10	Image Decryption Algorithm	09
11	Encryption Source Code	10-17
12	Decryption Source Code	17-24
13	Input/Output with Images	24-25
14	Histogram Analysis	25-26
15	Advantages of Logistic Map Combined with DNA Encryption	26
16	Conclusion	27
17	References	27

ACKNOWLEDGEMENT

It is a pleasure to acknowledge the assistance received from several individuals during the preparation of this project. We take this opportunity to extend our gratitude for all of them.

We are expressing our sincere gratitude and regards to our respected project coordinator Prof. Soumya Pratik Saha who has helped us not only from a very novice stage but also gave his frequent valuable advice and master suggestions every stage lead to systematic approach in completing this project without which this could not take its shape.

We would like to express our gratitude to the entire Computer Science department of our college who helped us in their own way whenever needed.

Lastly, it is our parents without whose help and courage it would have not been possible.

Submitted by :

1. DEEPANJAN BISWAS

Reg. No. 1011911401044, Roll No. 622110100981

2. ISHA SHARMA

Reg. No.1011921401055, Roll No. 622210100989



Acharya Prafulla Chandra College

New Barrackpore, Kolkata – 700131, West Bengal

CERTIFICATE

This is to certify that the project entitled '**Encryption of Image Using Logistic Map and DNA Encryption**' is done by *Deepanjan Biswas* and *Isha Sharma* under the guidance and supervision of Prof. Soumya Pratik Saha, for the partial fulfillment of degree in Bachelor of Science in Computer Science from Acharya Prafulla Chandra College affiliated to West Bengal State University.

.....
(Project Guide)

ABSTRACT:

Image encryption has become an increasingly crucial task in today's society, as technology advances on a daily basis. It is necessary to safeguard the photos, encrypt them, and then transport them through a channel. As the fast-growing era gets more prone to using pictures during talks, cryptography becomes a crucial instrument for ciphering the information before sending it across a media. Because the transmission method may be unsafe, the data being conveyed must be secure enough that only the legitimate recipient can decipher the image's original meaning. Furthermore, before encrypting a picture, it should be checked for general compatibility, and that image might be pre-processed using current image processing techniques. However, the suggested system is a study of how cryptography frequently proves to be a useful security tool, and it includes DNA sequence encoding as an extra layer. There are four modules in it. The first module involves creating a secret key with the use of a chaotic logistic map and analyzing the chaotic system's behavior. The quality of the image is then made more complicated by using DNA ideas in the following step, which is followed by general encryption to produce an encrypted image. Decoding the picture to obtain the original simple image would be the final step. The findings clearly suggest that using an encryption technique based on chaotic logistic mapping plus DNA encoding produces better results than using alone chaotic logistic mapping.

INTRODUCTION:

With the development of science, technology, and society, the computer industry, in which a small branch of digital images applications has become increasingly pervasive, has come to occupy a dominant position worldwide. Digital images have become one of the most popular media types and are now used extensively in various fields such as politics, economics, defense, and education [1]. However, because of the open nature of networks, image transmission security is subject to potential threats. In some fields, such as military affairs, commerce, and medical treatment, digital images also need to meet the highest requirements of confidentiality [2]. Consequently, image encryption technology has become an effective way to protect images being transmitted. Various common image encryption algorithms are available, including text encryption technology, SCAN language based encryption technology, quad tree image encryption technology, vector quantization encryption technology (VQ), encryption technology based on pseudorandom sequences, encryption technology based on the “key image” chaotic encryption technology, and image encryption technology based on DNA computing [3–10]. Recently, chaotic encryption technology has been attracting increasing attention. Chaos is an inner-class random process of nonlinear systems performance and is very sensitive to initial values, thus resulting in unpredictable results. The benefits of chaotic encryption technology include simple implementation, robustness, fast encryption, and high security [11]. However, although chaotic encryption technology has many advantages, it also has a number of deficiencies. For example, at present, most chaotic encryption algorithms confuse the single image pixel value or location, but the utilization of only one of the two strategies does not ensure high security for the image [12], and thus it is easy for attackers to crack an encrypted image by simply using the pixel comparison method. In 1994, Adleman first introduced DNA computing into the encryption field, which created a new stage of information processing. DNA encryption is a new frontier and is presently at the forefront of international cryptography research [13, 14]. DNA molecules harness massive parallelism and have low energy consumption and high storage density [15, 16]. Therefore, image encryption algorithms based on DNA computing possess unique advantages that the traditional cryptographic algorithms do not have. However, using only DNA encoding to encrypt images is not secure. Therefore, we combine chaos encryption technology and image encryption based on DNA computing to solve the hidden insecurity problems existing when images are confused using the chaotic encryption technology. First, we confuse the digital image pixels using the chaotic encryption technology. We then diffuse the confused pixels using DNA encoding. The diffusion process is also applied to the chaotic encryption technology and, finally, we obtain the encryption result. In summary, our study successfully combines chaotic encryption technology and DNA coding techniques in a method that has been verified via a large number of experiments and security analyses to prove the security and rationality of the algorithm.

Logistic map:

The logistic chaotic map is a polynomial map of depth two. The mathematical definition of this map is as follows

$$X_{n+1} = \mu X_n(1 - X_n). \quad (1)$$

Therefore, our algorithm confuses the pixels using the logistic map and suppose that the size of the original grayscale image I is $M \times N$. In summary, the main idea is as follows.

Step 1. Suppose two arrays, R and Cl , are, respectively, used to record the rows and columns of an image:

$$R = \{1, 2, \dots, M\} \quad \text{and} \quad Cl = \{1, 2, \dots, N\} \quad (2)$$

Step 2. Generate two pseudorandom sequences A and B , with respective sizes m and n , using the logistic map:

$$A = \{a_1, a_2, \dots, a_m\}, \quad \text{and} \quad B = \{a_1, a_2, \dots, a_n\}. \quad (3)$$

Step 3. Arrange the sequences A and B in descending order and record their locations. Thus, we can obtain the descending indexes, Index 1 and Index 2, of this pseudorandom sequence:

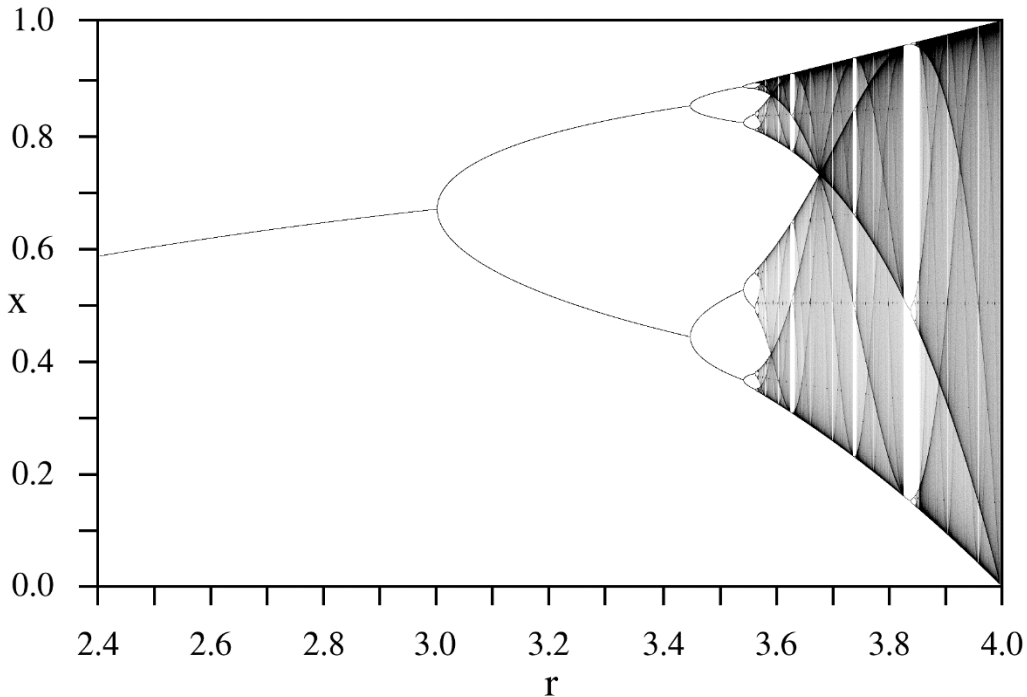
$$\text{Index 1} = \{i_1, i_2, \dots, i_m\} \quad \text{and} \quad \text{Index 2} = \{j_1, j_2, \dots, j_n\}. \quad (4)$$

The primary objective of this step is to find the index of the largest number from the sequence with size m and then store it in i_1 . Next, we find the index of the second largest number and store it in i_2 . We repeat this process until the descending sequence indexes are all stored in array Index 1. Similarly, we can obtain the index array Index 2 by arranging the sequence B in descending order and taking its index.

Step 4. According to indexes Index 1 and Index 2, we get new row R' and column C' . We can then obtain new arrays of the exchanged row R' and column C' and can confuse the image pixels.

Here μ is known as bifurcation parameter having range from 0 to 4.

$p(0)$ is the initial value $0 < p < 1$ and $\{p_1, p_2, p_3, \dots, p_n\}$ sequence elements are generated as per the equation 1.



DNA Encryption:

DNA sequencing is the process used to map the nucleotide sequence forming a strand of DNA. Four bases, adenine (A), thymine (T), guanine (G), and cytosine (C) form the building blocks of genetic code. "A" binds with "T" and "G" binds with "C" [17]. We know that every digital image pixel can be expressed by 8-bit binary numbers. If we use the four deoxyribonucleotides "A," "T," "G," and "C" to represent the binary numbers "00," "11," "01," and "10," respectively, then each pixel can be encoded into a string of nucleotides. For example, the gray value of a digital image pixel is 228, and the binary corresponding to this value is "11100100." According to the above rules, the string of nucleotides that corresponds to this binary is "TCGA."

We assume that the size of the original grayscale image I is $M \times N$, transform I into a binary matrix I' , and then randomly select one of the eight combinations of coding DNA to encode I' . The coded matrix is called I'' . Finally, I'' is converted into a one-dimensional sequence X , which can be expressed as follows:

$$X = \{x_1, x_2, x_3, \dots, x_{4MN}\}, x_i \in \{A, T, G, C\}. \quad (1)$$

We out of all possibilities of allotting the binary digits the four deoxyribonucleotides "A," "T," "G," and "C", we chose 8 ways according to the given table below:

Table 1: Assigning deoxyribonucleotides to the binary bits

1	2	3	4	5	6	7	8
00-A	00-A	00-C	00-C	00-G	00-G	00-T	00-T
01-C	01-G	01-A	01-T	01-A	01-T	01-C	01-G
10-G	10-C	10-T	10-A	10-T	10-A	10-G	10-C
11-T	11-T	11-G	11-G	11-C	11-C	11-A	11-A

After the allotment of the deoxyribonucleotides, we now create an extract 4 deoxyribonucleotides from the list X and exchange the positions of each, in the following steps:

Step 1: we extract the middle two letters and put them in the front and the mean position letters at the end. For example if the group of 4 letters is: "TCAG", the combination becomes "CATG"

Step 2: we now create an array of binary codes between the decimal digits of 0 to 15 and arrange them with the help of the logistic maps rule, explained above.

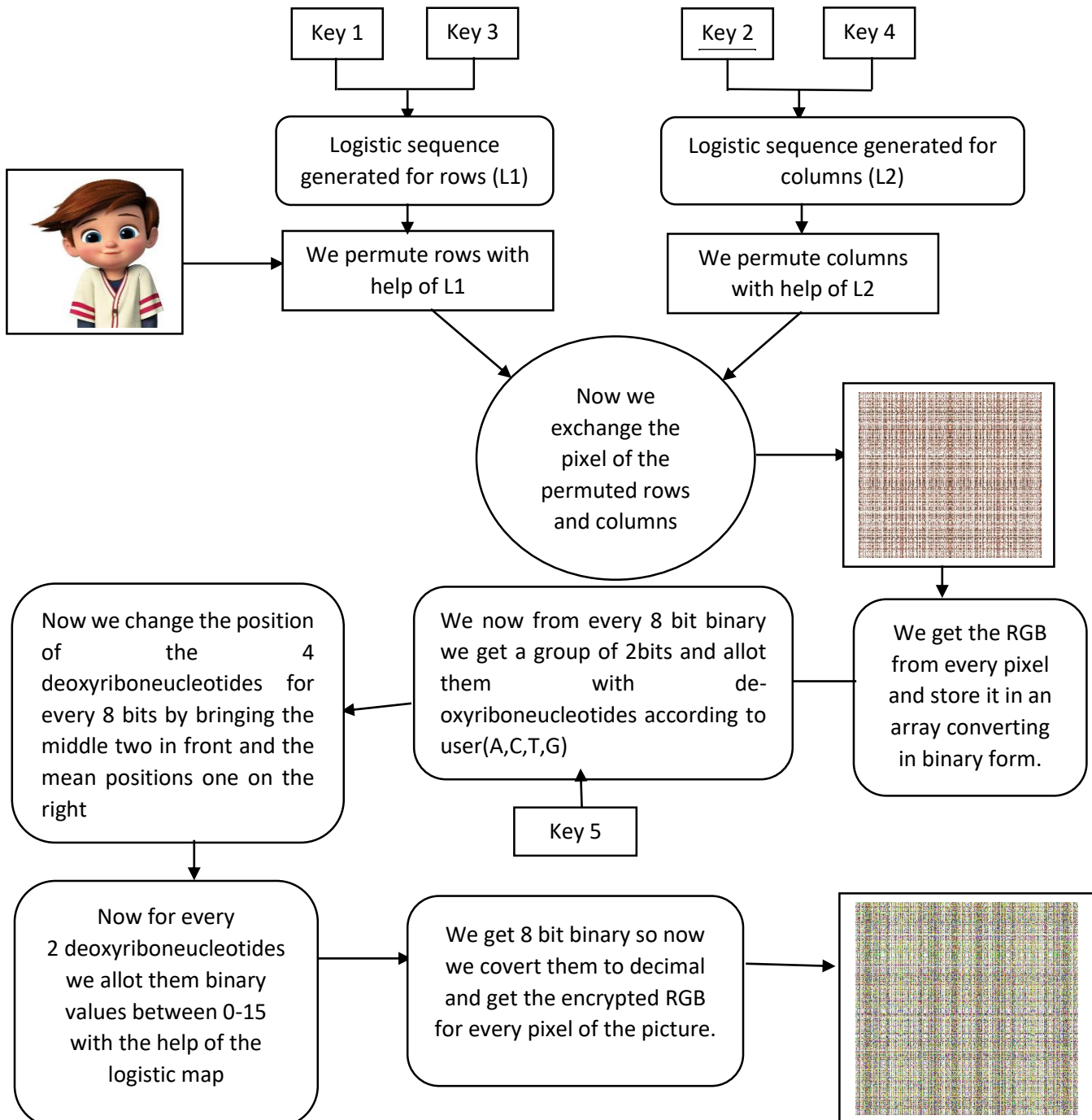
Step 3: let us say if we name the binary elements array Y . Then from 4 letters arranged above in step 1, we extract 2 letters every time For example: from the combination of "CATG", firstly we take "CA" and allot them binary bits of 4 digits according to the table 2. And then we extract "TG" and allot them binary bits of 4 digits according to table 2. We can have possible 16 arrangements between the letters.

Table 2: Allotting the group of two letters with 4-digit array given in the list Y .

	A	C	T	G
A	AA Y[0]	AC Y[1]	AT Y[2]	AG Y[3]
C	CA Y[4]	CC Y[5]	CT Y[6]	CG Y[7]
T	TA Y[8]	TC Y[9]	TT Y[10]	TG Y[11]
G	GA Y[12]	GC Y[13]	GT Y[14]	GG Y[15]

Step 4: For every four letters now we get 4+4=8bits of binary digits. Now we get an array of binary bits each of 8bits

How It Works:



Tools Used:

- Python as the coding base.
- We used the Pillow package as PIL to import and work in the image
- We used the math package

Image Encryption Algorithm. :

If we suppose that the size of the original grayscale image I is $M \times N$, the encryption steps are as follows:

Input: image I , the initial values of the logistic map x and $x1$, the parameters μ and $\mu1$, the option for DNA encryption between 1-8.

Output: Encrypted Image

Step 1. Convert the original image I into a two dimensional matrix called I . Let two arrays be named R and C . These are, respectively, used to record the rows and columns of image I .

Step 2. Generate the two one-dimensional descending index sequences using (1) of the logistic mapping, which is used to, respectively, exchange rows and columns of matrix I . Thus, we can obtain a confused image I' .

Step 3. From the image I' , now we extract each pixel values and store them in one dimensional matrix X which will have a length of $4 \times M \times N$.

Step 4: Then, generate a random integer $r1$ from one to eight, which is used to decide which DNA encoding rule (Table 1) should be used. Convert X into a DNA encoding matrix with $M \times N$ rows and four columns. Finally, convert this matrix into a one-dimensional DNA coding sequence X' , whose size is $M \times N \times 4$. Now we apply the DNA Algorithm on the matrix X' , and generate a new matrix X'' which will have the binary numbers with the length of 8bit each and total matrix length of $4 \times M \times N$.

Step 5: Convert the sequence X'' into a decimal dimensional matrix III with M rows and N columns. Finally, convert the dimensional matrix III into the encryption image III and output the encryption image.

Image Decryption Algorithm :

Step 1: Firstly open the encrypted image and convert the pixels into a single binary array and group into 4 bit each and according to the given logistic values we allot the same (A,C,T,G)s to the binary

Step 2: we convert the De oxyriboneucleotides into the given choice of the key 5 .

Step 3: now we re arrange the rows and columns using the key1, key2, key3, key4 and reverse algorithm of logistic map

Step 4: We allot back the interchanged pixels back to the original positions

Step 5: we save the decrypted Image

Source Code for Encryption:

```

from PIL import Image
import math
# Import an image from directory:

main_image = Image.open("E:/photo.jpg")
# Extracting pixel map:
main_image.show()

pixel_map = main_image.load()
# Extracting the width and height of the image:

width, height = main_image.size
w=[]
for i in range(width):
    w.append(i)
h=[]
for i in range(height):
    h.append(i)

d=[]
k=0
for i in range(width):
    c=[]
    k=[]
    for j in range(height):

        # getting the RGB pixel value.
        r, g, b= main_image.getpixel((i, j))
        c.append(r)
        c.append(g)
        c.append(b)
        k.append(c)
    d.append(k)
#print(len(d))
meu=float(input("Enter the value of key 1, give value between 3.5 and 4:  "))
while(meu<=3.5 or meu>4):
    meu=float(input("You entered wrong key enter value between 3.5 and 4:  "))
meu1=float(input("Enter the value of key 2, give value between 3.5 and 4:  "))
while(meu1<=3.5 or meu1>4):
    meu1=float(input("You entered wrong key enter value between 3.5 and 4:  "))
x=float(input("Enter the value of key 3, give value greater than 0 and less than 1:  "))
while(x<0 or x>1):
    x=float("Entered wrong value enter greater than 0 and less than 1:  ")
x1=float(input("Enter the value of key 4, give value greater than 0 and less than 1:  "))
while(x<0 or x>1):

```

```

x1=float("Entered wrong value enter greater than 0 and less than 1: ")
c=[]
for i in range(width):
    xn=meu*x*(1-x)
    c.append(xn)
    x=xn

#storing meu2 and xf for DNA cryptography
meu2=meu
xf=x1
c1=[]
for i in range(height):
    xn=meu1*x1*(1-x1)
    c1.append(xn)
    x1=xn
#print(len(c),len(c1))
#setting the rows with the help of logistic map array created in c
for i in range(len(w)):
    m=i
    for j in range(i+1,len(w)):
        if(c[m]<c[j]):
            m=j
    c[i],c[m]=c[m],c[i]
    w[i],w[m]=w[m],w[i]

#setting the coloumn with the use of array c1
for i in range(len(h)):
    m=i
    for j in range(i+1,len(h)):
        if(c1[m]<c1[j]):
            m=j
    c1[i],c1[m]=c1[m],c1[i]
    h[i],h[m]=h[m],h[i]

p=[]
for i in w:
    for j in h:

        # getting the RGB pixel value.

        r, g, b= main_image.getpixel((i, j))
        p.append(r)
        p.append(g)
        p.append(b)
count=0
for i in range(len(w)):
    for j in range(len(h)):
        r=p[count]
        g=p[count+1]
        b=p[count+2]

```

```

count=count+3
pixel_map[i,j]=(r,g,b)

main_image.save("E:/encryptedIMGnew11.png", format="png")

main_image = Image.open("E:/encryptedIMGnew11.png")

# Extracting pixel map:

pixel_map = main_image.load()
# Extracting the width and height of the image:

width, height = main_image.size

meu2=meu
xf=x1
c=[]
for i in range(width):
    for j in range(height):

        # getting the RGB pixel value.

        r, g, b= main_image.getpixel((i, j))
        c1=r
        c2=g
        c3=b
        c.append(c1)
        c.append(c2)
        c.append(c3)
#storing the pixels value in binary form in d array
d=[]
for a in c:
    b=bin(a).replace('0b', '')
    while(len(b)<8):
        b='0'+b
    d.append(b)

#Alloting the 2-bit binary a letter by the key given by user
choice=int(input("Enter the key 5, value should be between 1-8: "))
while(choice>8 or choice==0):
    choice=int(input("You entered wrong key enter between 1-8: "))
print()
print("All your keys are accepted!!!")

st=[]
if(choice==1):
    for i in d:
        for j in range(0,8,2):
            tmp=i[j:j+2]
            if(tmp=='00'):

```

```
        st.append('A')
    elif(tmp=='11'):
        st.append('T')
    elif(tmp=='01'):
        st.append('C')
    elif(tmp=='10'):
        st.append('G')
elif(choice==2):
    for i in d:
        for j in range(0,8,2):
            tmp=i[j:j+2]
            if(tmp=='00'):
                st.append('A')
            elif(tmp=='01'):
                st.append('G')
            elif(tmp=='10'):
                st.append('C')
            elif(tmp=='11'):
                st.append('T')
elif(choice==3):
    for i in d:
        for j in range(0,8,2):
            tmp=i[j:j+2]
            if(tmp=='00'):
                st.append('C')
            elif(tmp=='01'):
                st.append('A')
            elif(tmp=='10'):
                st.append('T')
            elif(tmp=='11'):
                st.append('G')
elif(choice==4):
    for i in d:
        for j in range(0,8,2):
            tmp=i[j:j+2]
            if(tmp=='00'):
                st.append('C')
            elif(tmp=='01'):
                st.append('T')
            elif(tmp=='10'):
                st.append('A')
            elif(tmp=='11'):
                st.append('G')
elif(choice==5):
    for i in d:
        for j in range(0,8,2):
            tmp=i[j:j+2]
            if(tmp=='00'):
                st.append('G')
            elif(tmp=='01'):
                st.append('A')
```

```
        elif(tmp=='10'):
            st.append('T')
        elif(tmp=='11'):
            st.append('C')
    elif(choice==6):
        for i in d:
            for j in range(0,8,2):
                tmp=i[j:j+2]
                if(tmp=='00'):
                    st.append('G')
                elif(tmp=='01'):
                    st.append('T')
                elif(tmp=='10'):
                    st.append('A')
                elif(tmp=='11'):
                    st.append('C')
    elif(choice==7):
        for i in d:
            for j in range(0,8,2):
                tmp=i[j:j+2]
                if(tmp=='00'):
                    st.append('T')
                elif(tmp=='01'):
                    st.append('C')
                elif(tmp=='10'):
                    st.append('G')
                elif(tmp=='11'):
                    st.append('A')
    elif(choice==8):
        for i in d:
            for j in range(0,8,2):
                tmp=i[j:j+2]
                if(tmp=='00'):
                    st.append('T')
                elif(tmp=='01'):
                    st.append('G')
                elif(tmp=='10'):
                    st.append('C')
                elif(tmp=='11'):
                    st.append('A')

    alot=[]
    for i in range(16):
        s=bin(i).replace("0b","")
        x=s[::-1]
        while(len(x)<4):
            x=x+"0"
        s=x[::-1]
        alot.append(str(s))
```

```

tmp1=[]
for i in range(16):
    xn=meu2*(1-xf)
    tmp1.append(xf)
    xf=xn
for i in range(len(alot)):
    m=i
    for j in range(i+1,len(alot)):
        if(tmp1[m]<tmp1[j]):
            m=j
    tmp1[i],tmp1[m]=tmp1[m],tmp1[i]
    alot[i],alot[m]=alot[m],alot[i]
#print(alot)
c=[]
d=[]
tmp=[]
a=0
for i in range(0,len(st),4):
    c.append(st[i+1])
    c.append(st[i+2])
    d.append(st[i])
    d.append(st[i+3])
    if(c[a]=="A" and c[a+1]=="A"):
        tmp.append(alot[0])
    elif(c[a]=="A" and c[a+1]=="C"):
        tmp.append(alot[1])
    elif(c[a]=="A" and c[a+1]=="T"):
        tmp.append(alot[2])
    elif(c[a]=="A" and c[a+1]=="G"):
        tmp.append(alot[3])
    elif(c[a]=="C" and c[a+1]=="A"):
        tmp.append(alot[4])
    elif(c[a]=="C" and c[a+1]=="C"):
        tmp.append(alot[5])
    elif(c[a]=="C" and c[a+1]=="T"):
        tmp.append(alot[6])
    elif(c[a]=="C" and c[a+1]=="G"):
        tmp.append(alot[7]);
    elif(c[a]=="T" and c[a+1]=="A"):
        tmp.append(alot[8]);
    elif(c[a]=="T" and c[a+1]=="C"):
        tmp.append(alot[9]);
    elif(c[a]=="T" and c[a+1]=="T"):
        tmp.append(alot[10]);
    elif(c[a]=="T" and c[a+1]=="G"):
        tmp.append(alot[11]);
    elif(c[a]=="G" and c[a+1]=="A"):
        tmp.append(alot[12]);
    elif(c[a]=="G" and c[a+1]=="C"):
        tmp.append(alot[13]);
    elif(c[a]=="G" and c[a+1]=="T"):

```



```

    tmp.append(alot[14]);
elif(c[a]=="G" and c[a+1]=="G"):
    tmp.append(alot[15]);
c=[]
if(d[a]=="A" and d[a+1]=="A"):
    tmp.append(alot[0])
elif(d[a]=="A" and d[a+1]=="C"):
    tmp.append(alot[1])
elif(d[a]=="A" and d[a+1]=="T"):
    tmp.append(alot[2])
elif(d[a]=="A" and d[a+1]=="G"):
    tmp.append(alot[3])
elif(d[a]=="C" and d[a+1]=="A"):
    tmp.append(alot[4])
elif(d[a]=="C" and d[a+1]=="C"):
    tmp.append(alot[5])
elif(d[a]=="C" and d[a+1]=="T"):
    tmp.append(alot[6])
elif(d[a]=="C" and d[a+1]=="G"):
    tmp.append(alot[7]);
elif(d[a]=="T" and d[a+1]=="A"):
    tmp.append(alot[8]);
elif(d[a]=="T" and d[a+1]=="C"):
    tmp.append(alot[9]);
elif(d[a]=="T" and d[a+1]=="T"):
    tmp.append(alot[10]);
elif(d[a]=="T" and d[a+1]=="G"):
    tmp.append(alot[11]);
elif(d[a]=="G" and d[a+1]=="A"):
    tmp.append(alot[12]);
elif(d[a]=="G" and d[a+1]=="C"):
    tmp.append(alot[13]);
elif(d[a]=="G" and d[a+1]=="T"):
    tmp.append(alot[14]);
elif(d[a]=="G" and d[a+1]=="G"):
    tmp.append(alot[15]);
d=[]

```

#converting the encoded binary into a character

```

m=[]
for i in range(0,len(tmp),2):
    s=tmp[i]+tmp[i+1]
    cal=0
    s=int(s)
    c=0
    while(s>0):
        d=s%10
        cal=cal+int(math.pow(2,c))*d
        s=s//10
        c=c+1
    m.append(cal)

```

```

count=0
for i in range(width):
    for j in range(height):

        # getting the RGB pixel value.

        r=m[count]
        g=m[count+1]
        b=m[count+2]
        count=count+3
        pixel_map[i, j] = (int(r), int(g), int(b))
        #print(pixel_map[i,j])
main_image.save("E:/encryptedIMG.png", format="png")
main_image.show()
print("ENCRYPTION DONE.. ♥😊😁")

```

Decryption Source Code:

```

from PIL import Image
import math
# Import an image from directory:
main_image = Image.open("E:/encryptedIMG.png")
#input_image1 = Image.open("E:/picture.png")
main_image.show()
# Extracting pixel map:

pixel_map = main_image.load()
# Extracting the width and height of the image:
width, height = main_image.size
meu=float(input("Enter the value of key 1, give value between 3.5 and 4:  "))
while(meu<=3.5 or meu>4):
    meu=float(input("You entered wrong key enter value between 3.5 and 4:  "))
meu1=float(input("Enter the value of key 2, give value between 3.5 and 4:  "))
while(meu1<=3.5 or meu1>4):
    meu1=float(input("You entered wrong key enter value between 3.5 and 4:  "))
x=float(input("Enter the value of key 3, give value greater than 0 and less than 1:  "))
while(x<0 or x>1):
    x=float("Entered wrong value enter greater than 0 and less than 1:  ")
x1=float(input("Enter the value of key 4, give value greater than 0 and less than 1:  "))
while(x<0 or x>1):
    x1=float("Entered wrong value enter greater than 0 and less than 1:  ")

#print(width,height)
c=[]
for i in range(width):
    for j in range(height):

        # getting the RGB pixel value.

```

```

    r, g, b= main_image.getpixel((i, j))
    c.append(r)
    c.append(g)
    c.append(b)
d=[]
for a in c:
    b=bin(a).replace('0b', '')
    while(len(b)<8):
        b='0'+b
    d.append(b)
a=[]
b=[]
for i in d:
    a.append(i[:4])
    b.append((i[4:8]))

alot=[]
for i in range(16):
    s=bin(i).replace("0b", "")
    x8=s[::-1]
    while(len(x8)<4):
        x8=x8+"0"
    s=x8[::-1]
    alot.append(str(s))

tmp1=[]
xf=x1
meu2=meu
for i in range(16):
    xn=meu2*(1-xf)
    tmp1.append(xf)
    xf=xn
for i in range(len(alot)):
    m=i
    for j in range(i+1,len(alot)):
        if(tmp1[m]<tmp1[j]):
            m=j
    tmp1[i],tmp1[m]=tmp1[m],tmp1[i]
    alot[i],alot[m]=alot[m],alot[i]

c=[]
d=[]
for i in a:
    if(i==alot[0]):
        c.append("AA")
    elif(i==alot[1]):
        c.append("AC")
    elif(i==alot[2]):
        c.append("AT")
    elif(i==alot[3]):
        c.append("AG")

```

```
elif(i==alot[4]):
    c.append("CA")
elif(i==alot[5]):
    c.append("CC")
elif(i==alot[6]):
    c.append("CT")
elif(i==alot[7]):
    c.append("CG")
elif(i==alot[8]):
    c.append("TA")
elif(i==alot[9]):
    c.append("TC")
elif(i==alot[10]):
    c.append("TT")
elif(i==alot[11]):
    c.append("TG")
elif(i==alot[12]):
    c.append("GA")
elif(i==alot[13]):
    c.append("GC")
elif(i==alot[14]):
    c.append("GT")
elif(i==alot[15]):
    c.append("GG")
for i in b:
    if(i==alot[0]):
        d.append("AA")
    elif(i==alot[1]):
        d.append("AC")
    elif(i==alot[2]):
        d.append("AT")
    elif(i==alot[3]):
        d.append("AG")
    elif(i==alot[4]):
        d.append("CA")
    elif(i==alot[5]):
        d.append("CC")
    elif(i==alot[6]):
        d.append("CT")
    elif(i==alot[7]):
        d.append("CG")
    elif(i==alot[8]):
        d.append("TA")
    elif(i==alot[9]):
        d.append("TC")
    elif(i==alot[10]):
        d.append("TT")
    elif(i==alot[11]):
        d.append("TG")
    elif(i==alot[12]):
        d.append("GA")
```

```

elif(i==alot[13]):
    d.append("GC")
elif(i==alot[14]):
    d.append("GT")
elif(i==alot[15]):
    d.append("GG")
#print(c)
#print(d)
k=[]
s=""
for i in range(0,len(c)):
    s=d[i][0:1]+c[i][0:1]+c[i][1:2]+d[i][1:2]
    k.append(s)
    s=""
#print(k)
choice=int(input("Enter the key 5, value should be between 1-8: "))
while(choice>8 or choice==0):
    choice=int(input("You entered wrong key enter between 1-8: "))

tmp=[]
if(choice==1):
    for i in k:
        for j in range(0,4):
            if(i[j:j+1]=="A"):
                s=s+"00"
            elif(i[j:j+1]=="C"):
                s=s+"01"
            elif(i[j:j+1]=="G"):
                s=s+"10"
            elif(i[j:j+1]=="T"):
                s=s+"11"
        tmp.append(s)
    s=""
if(choice==2):
    for i in k:
        for j in range(0,4):
            if(i[j:j+1]=="A"):
                s=s+"00"
            elif(i[j:j+1]=="G"):
                s=s+"01"
            elif(i[j:j+1]=="C"):
                s=s+"10"
            elif(i[j:j+1]=="T"):
                s=s+"11"
        tmp.append(s)
    s=""
if(choice==3):
    for i in k:
        for j in range(0,4):
            if(i[j:j+1]=="C"):
                s=s+"00"

```

```
        elif(i[j:j+1]=="A"):
            s=s+"01"
        elif(i[j:j+1]=="T"):
            s=s+"10"
        elif(i[j:j+1]=="G"):
            s=s+"11"
    tmp.append(s)
    s=""
if(choice==4):
    for i in k:
        for j in range(0,4):
            if(i[j:j+1]=="C"):
                s=s+"00"
            elif(i[j:j+1]=="T"):
                s=s+"01"
            elif(i[j:j+1]=="A"):
                s=s+"10"
            elif(i[j:j+1]=="G"):
                s=s+"11"
        tmp.append(s)
        s=""
if(choice==5):
    for i in k:
        for j in range(0,4):
            if(i[j:j+1]=="G"):
                s=s+"00"
            elif(i[j:j+1]=="A"):
                s=s+"01"
            elif(i[j:j+1]=="T"):
                s=s+"10"
            elif(i[j:j+1]=="C"):
                s=s+"11"
        tmp.append(s)
        s=""
if(choice==6):
    for i in k:
        for j in range(0,4):
            if(i[j:j+1]=="G"):
                s=s+"00"
            elif(i[j:j+1]=="T"):
                s=s+"01"
            elif(i[j:j+1]=="A"):
                s=s+"10"
            elif(i[j:j+1]=="C"):
                s=s+"11"
        tmp.append(s)
        s=""
if(choice==7):
    for i in k:
        for j in range(0,4):
            if(i[j:j+1]=="T"):
```

```

        s=s+"00"
    elif(i[j:j+1]=="C"):
        s=s+"01"
    elif(i[j:j+1]=="G"):
        s=s+"10"
    elif(i[j:j+1]=="A"):
        s=s+"11"
    tmp.append(s)
    s=""
if(choice==8):
    for i in k:
        for j in range(0,4):
            if(i[j:j+1]=="T"):
                s=s+"00"
            elif(i[j:j+1]=="G"):
                s=s+"01"
            elif(i[j:j+1]=="C"):
                s=s+"10"
            elif(i[j:j+1]=="A"):
                s=s+"11"
        tmp.append(s)
        s=""
#print(len(tmp))
tmp1=[]
for i in tmp:
    s=int(i)
    cal=0
    c=0
    while(s>0):
        d=s%10
        cal=cal+int(math.pow(2,c))*d
        s=s//10
        c=c+1
    tmp1.append(cal)
#print(len(tmp1)//3)
#print(len(tmp1))
count=0
for i in range(width):
    for j in range(height):

        # getting the RGB pixel value
        r=tmp1[count]
        g=tmp1[count+1]
        b=tmp1[count+2]
        count=count+3
    # r1,g1,b1=input_image1.getpixel((i,j))
    # Apply formula of xor:
    c1=r
    c2=g
    c3=b
    #print(c1,"",c2,"",c3)

```

```

    # setting the pixel value.
    pixel_map[i, j] = (int(c1), int(c2), int(c3))
# Saving the final output
# as "decrypted final.png":

main_image.save("E:/deryptedIMGnew.png", format="png")

main_image = Image.open("E:/deryptedIMGnew.png")
# Extracting pixel map:
#main_image.show()

pixel_map = main_image.load()
# Extracting the width and height of the image:

width, height = main_image.size

c=[]
d=[]
m=0
for i in range(width):
    xn=meu*x*(1-x)
    c.append(xn)
    d.append(xn)
    x=xn

m1=0
c1=[]
d1=[]
for i in range(height):
    xn=meu1*x1*(1-x1)
    c1.append(xn)
    d1.append(xn)
    x1=xn
w=[]
h=[]
for i in range(len(c)):
    m=i
    for j in range(i+1,len(c)):
        if(c[m]<c[j]):
            m=j
    c[i],c[m]=c[m],c[i]

for i in range(len(c1)):
    m=i
    for j in range(i+1,len(c1)):
        if(c1[m]<c1[j]):
            m=j
    c1[i],c1[m]=c1[m],c1[i]

for i in range(len(c)):
    for j in range(len(d)):

```



```

    if(c[i]==d[j]):
        w.append(j)


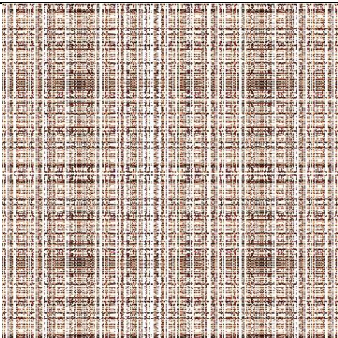
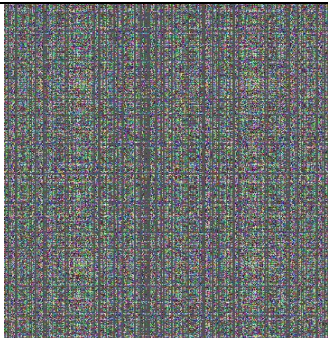
for i in range(len(c1)):
    for j in range(len(d1)):
        if(c1[i]==d1[j]):
            h.append(j)

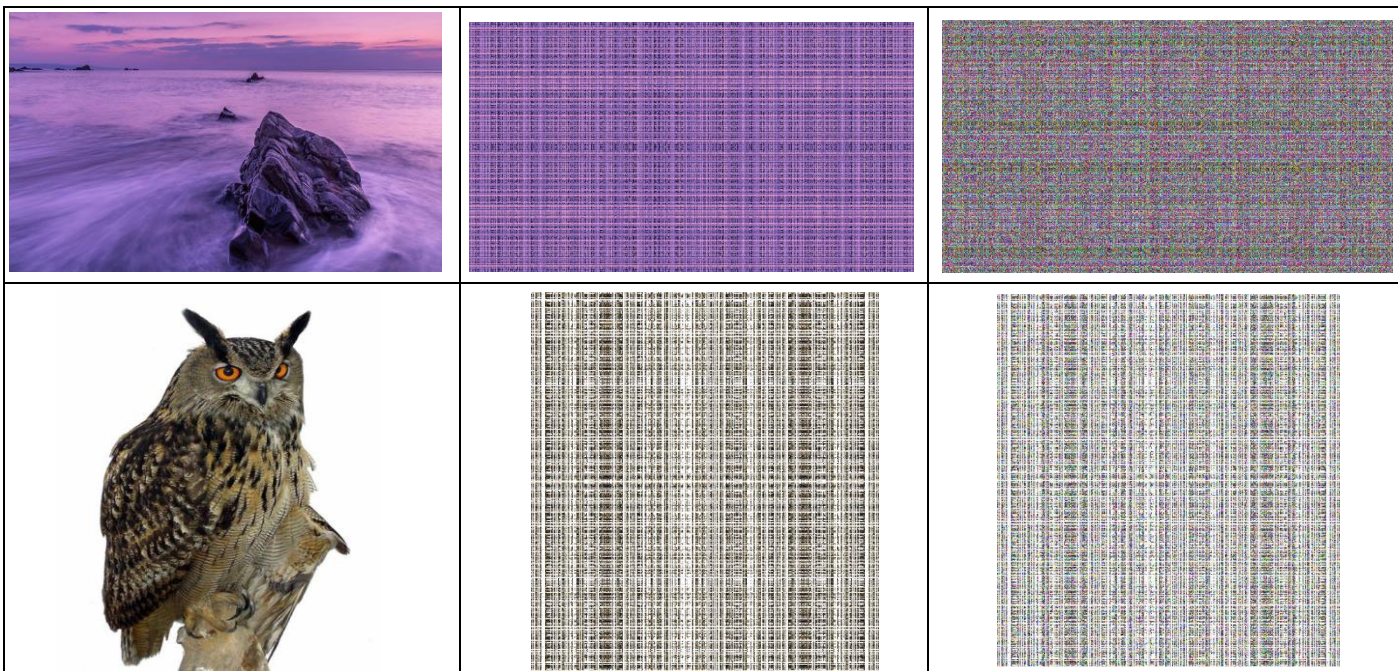
c=0
p=[]
for i in range(len(w)):
    for j in range(len(h)):
        r,g,b=main_image.getpixel((i,j))
        p.append(r)
        p.append(g)
        p.append(b)
count=0
for i in w:
    for j in h:
        r=p[count]
        g=p[count+1]
        b=p[count+2]
        count=count+3
        pixel_map[i,j]=(r,g,b)

main_image.save("E:/decryptedIMG.png", format="png")
main_image.show()
print("DECRYPTION DONE.. ❤️😊😁")

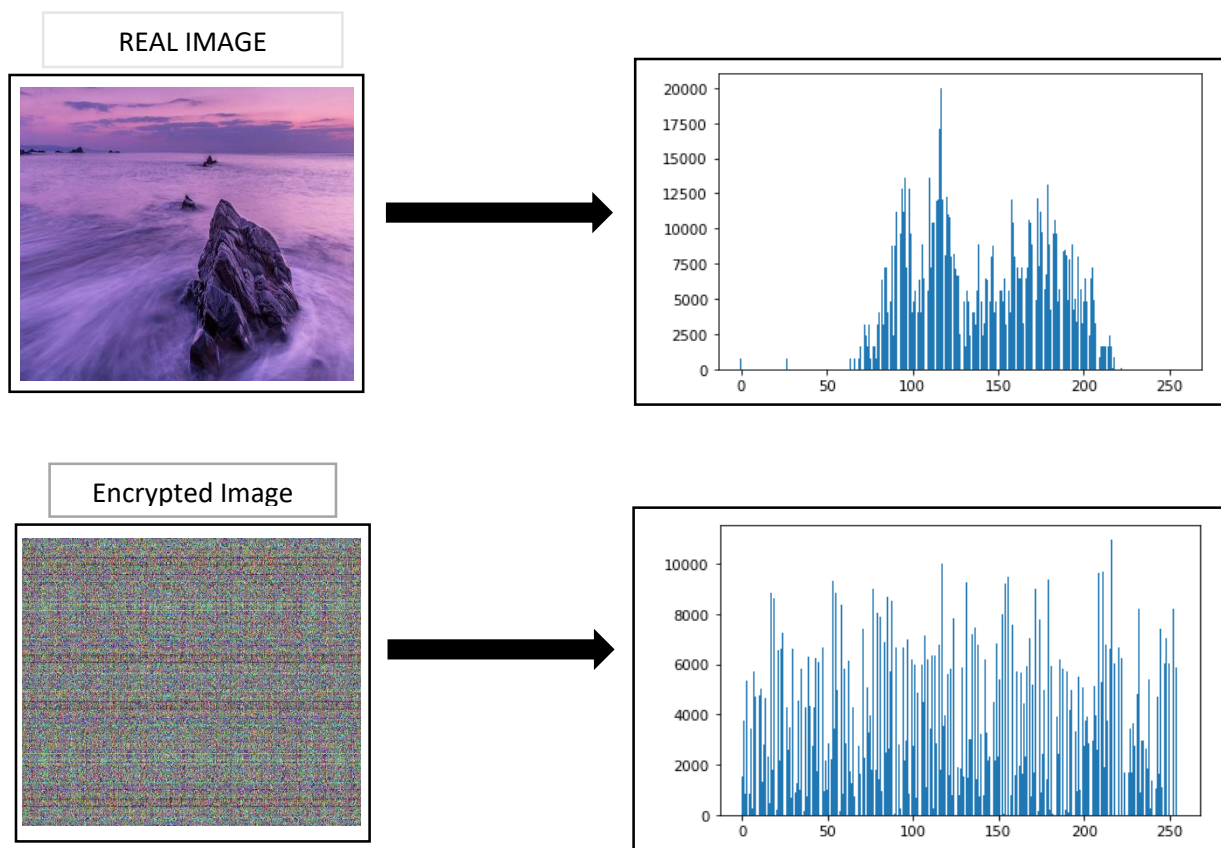
```

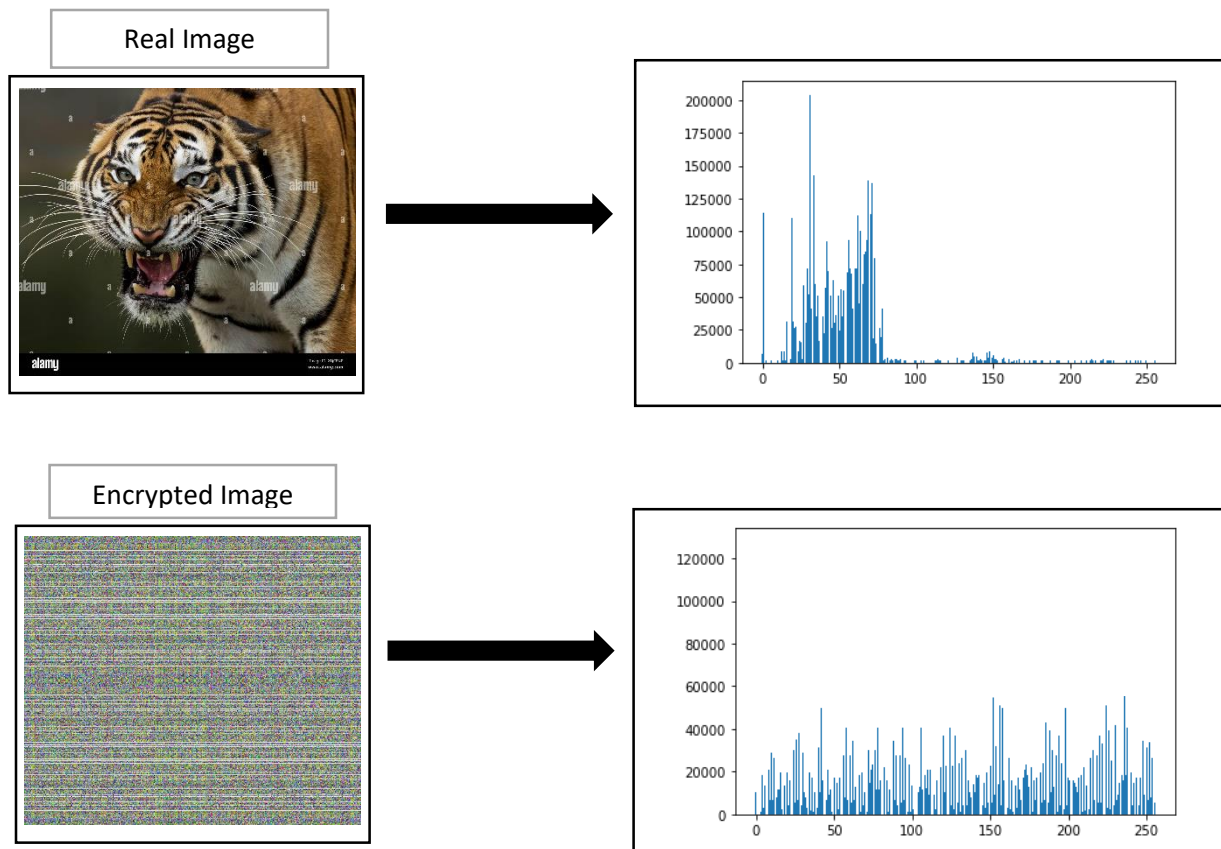
Input and Output With Images:

Input Image	Permuted rows and columns	Encrypted image
		



HISTOGRAM ANALYSIS:





Advantages of logistic map combined with DNA encryption:

The relative simplicity of the logistic map makes it a widely used point of entry into consideration of the concept of chaos. This algorithm is much quicker. It simply iterates recursively until there are as many values of XN as there are pixels in the original image. A rough description of chaos is that chaotic systems exhibit great sensitivity to initial conditions—a property of the logistic map for most values of r between about 3.57 and 4. The values of XN are scaled to be in $[0, 255]$ and the XOR is performed, as expected (Goa, Zhang, & Jiang, 2006). But the limitation of the logistic map turned out to be the range of r being $[3.5, 4]$ which led us to implement DNA encryption along with the logistic map. DNA molecule having the capacity to store, process and transmit information inspires the idea of DNA cryptography. One of the advantages of DNA computation is the massive parallelism of DNA molecules. In an, in vitro assay, about 10^{18} processors working in parallel can be easily handled. Because of this huge parallelism the trapdoor function, which is the basic security secret of most of traditional cryptosystems, can be solved in polynomial time. As the mathematical aspect of cryptography is being replaced by DNA chemistry in the domain of DNA cryptography, thus this technique is virtually unhackable by conventional methods as well as quantum computation.

CONCLUSION:

In this paper we proposed an efficient algorithm for image encryption. Firstly, we interchange the pixels randomly by interchanging the rows and the columns using the logistic map algorithm. Secondly we used the DNA encryption technology to change the pixel values of each pixel in a calculated form. Private keys are used by the user to encrypt the image which will be further needed to decode the image. The usage of two algorithms to encrypt the image makes it a strong encryption process.

REFERENCES:

1. Y. Liu, X. Tong, and S. Hu, "A family of new complex number chaotic maps based image encryption algorithm," *Signal Processing: Image Communication*, vol. 28, no. 10, pp. 1548–1559, 2013.
2. Y. Zhou, L. Bao, and C. L. P. Chen, "A new 1D chaotic system for image encryption," *Signal Processing*, vol. 97, pp. 172–182, 2014.
3. A. Bakhshandeh and Z. Eslami, "An authenticated image encryption scheme based on chaotic maps and memory cellular automata," *Optics and Lasers in Engineering*, vol. 51, no. 6, pp. 665–673, 2013.
4. W. Chen, C. Quan, and C. J. Tay, "Optical color image encryption based on Arnold transform and interference method," *Optics Communications*, vol. 282, no. 18, pp. 3680–3685, 2009.
5. R. Enayatifar, A. H. Abdullah, and M. Lee, "A weighted discrete imperialist competitive algorithm (WDICA) combined with chaotic map for image encryption," *Optics and Lasers in Engineering*, vol. 51, no. 9, pp. 1066–1077, 2013.
6. A. Kadir, A. Hamdulla, and W.-Q. Guo, "Color image encryption using skew tent map and hyper chaotic system of 6th-order CNN," *Optik*, vol. 125, no. 5, pp. 1671–1675, 2014.
7. S. Lian, "A block cipher based on chaotic neural networks," *Neurocomputing*, vol. 72, no. 4–6, pp. 1296–1301, 2009.
8. A. N. Pisarchik and M. Zanin, "Image encryption with chaotically coupled chaotic maps," *Physica D: Nonlinear Phenomena*, vol. 237, no. 20, pp. 2638–2648, 2008.
9. J. W. Yoon and H. Kim, "An image encryption scheme with a pseudorandom permutation based on chaotic maps," *Communications in Nonlinear Science and Numerical Simulation*, vol. 15, no. 12, pp. 3998–4006, 2010.
10. F. Zheng, X. J. Tian, J. Y. Song, and X. Y. Li, "Pseudo-random sequence generator based on the generalized Henon map," *The Journal of China Universities of Posts and Telecommunications*, vol. 15, no. 3, pp. 64–68, 2008.
11. S. Mazloom and A. M. Eftekhari-Moghadam, "Color image encryption based on coupled nonlinear chaotic map," *Chaos, Solitons and Fractals*, vol. 42, no. 3, pp. 1745–1754, 2009.
12. H. Li and Y. Wang, "Double-image encryption based on discrete fractional random transform and chaotic maps," *Optics and Lasers in Engineering*, vol. 49, no. 7, pp. 753–757, 2011.
13. Q. Zhang, L. Guo, and X. Wei, "Image encryption using DNA addition combining with chaotic maps," *Mathematical and Computer Modelling*, vol. 52, no. 11–12, pp. 2028–2035, 2010.
14. Q. Zhang, Q. Wang, and X. Wei, "A novel image encryption scheme based on DNA coding and multi-chaotic maps," *Advanced Science Letters*, vol. 3, no. 4, pp. 447–451, 2010.
15. S. H. Jiao and R. Goutte, "Code for encryption hiding data into genomic DNA of living organisms," in *Proceedings of the 9th International Conference on Signal Processing (ICSP '08)*, pp. 2166–2169, Beijing, China, October 2008.
16. A. Kanso and M. Ghebleh, "A novel image encryption algorithm based on a 3D chaotic map," *Communications in Nonlinear Science and Numerical Simulation*, vol. 17, no. 7, pp. 2943–2959, 2012.