APPENDIX

Appendix I:

Enhanced RC4 Algorithm java code:

```
package rc4;
import java.awt.Color;
import java.awt.image.BufferedImage;
import java.io.DataInputStream;
import java.io.File;
import java.io.IOException;
import javax.imageio.IIOImage;
import javax.imageio.ImageIO;
import javax.imageio.ImageWriteParam;
import javax.imageio.ImageWriter;
import javax.imageio.plugins.bmp.BMPImageWriteParam;
import javax.imageio.stream.ImageOutputStream;
import static rc4.RC4.s2;
public class RC4 {
public static int s[]=new int[256];
public static int s2[]=new int[256];
public static int t2;
public static void main(String[] args) throws IOException {
//Main }
public static int[] initKey( String key) {
 int i=0,j=0,temp=0;
 char keyc[]=key.toCharArray();
```

```
int keyi[]=new int[key.length()];
for(int a=0;a<key.length();a++) {
 keyi[a]=(int)keyc[a]; }
for( i=0;i<255;i++) {
  s[i]=i;
  k[i%128]=keyi[i%key.length()]; }
    return k;  }
public static BufferedImage Decrypt(String old_path) {
int i=0;
int j=0;
int t1;
try { old_img = ImageIO.read(new File(old_path));}
catch (Exception e) {  e.printStackTrace(); }
BufferedImage new_img = new BufferedImage( old_img.getWidth(),
old_img.getHeight(),
BufferedImage.TYPE_INT_RGB);
int n = new_img.getWidth();
int m = new_img.getHeight();
int red,green,blue,cr,cg,cb;
int rgb ;
for ( i = 0; i < n; ++i) {
     for ( j = 0; j < m; ++j) {
           rgb = old_img.getRGB(i,j);
           blue = (rgb)&0xFF;
           green = (rgb>>8)&0xFF;
           red = (rgb>>16)&0xFF;
           t1= permutate(i,j);
```

```
        cr=s[t1]^red;
        t1= permutate(i,j);
        cg=s[t1]^green;
        t1=permutate(i,j);
        cb=s[t1]^blue;
        int rgb1=new Color(cr, cg, cb).getRGB();
       new_img.setRGB(i, j, rgb1) }  }
return new_img; } //end
```

Appendix II:

Original image Histogram :

```
I = imread('flower.jpg');
figure,imshow(I)
R=imhist(I(:,:,1));
G=imhist(I(:,:,2));
B=imhist(I(:,:,3));
subplot(3,1,1),plot(R,'r')
subplot(3,1,2),plot(G,'g')
subplot(3,1,3),plot(B,'b')
```

Encrypted image Histogram:

```
I = imread('etest.jpg');
figure,imshow(I)
R=imhist(I(:,:,1));
G=imhist(I(:,:,2));
B=imhist(I(:,:,3));
subplot(3,1,1),plot(R,'r')
```

```
subplot(3,1,2),plot(G,'g')

subplot(3,1,3),plot(B,'b')
```

PSNR & MSE code:

```
InputImage=imread('flower.JPG');

ReconstructedImage=imread('dtest.JPG');

n=size(InputImage);

 M=n(1);

 N=n(2);

 MSE = sum(sum((InputImage-ReconstructedImage).^2))/(M*N);

PSNR = 10*log10(256*256/MSE);

 fprintf('\n MSE: %7.2f ', MSE);

 fprintf('\nPSNR: %9.7f dB', PSNR)
```

Diehard Tool:

Diehard tool work in different platform I implement it under DOS.

1- Cd diehard
2- Diehard.exe
3- Enter the input file.

```
C:\Users\devloper> cd ..

C:\Users>cd ..

C:\>cd diehard

C:\diehard>diehard.exe
      NOTE: Most of the tests in DIEHARD return a p-value, which
      should be uniform on [0,1) if the input file contains truly
      independent random bits.   Those p-values are obtained by
      p=F(X), where F is the assumed distribution of the sample
      random variable X---often normal. But that assumed F is just
      an asymptotic approximation, for which the fit will be worst
      in the tails. Thus you should not be surprised with
      occasional p-values near 0 or 1, such as .0012 or .9983.
      When a bit stream really FAILS BIG, you will get p's of 0 or
      1 to six or more places.  By all means, do not, as a
      Statistician might, think that a p < .025 or p> .975 means
      that the RNG has "failed the test at the .05 level".  Such
      p's happen among the hundreds that DIEHARD produces, even
      with good RNG's.  So keep in mind that " p happens".
 Enter filename (<=15 characters):
erc41.txt
 Enter name of output file (<=15 characters):
result
```

4- Select the choice of test

```
     HERE ARE YOUR CHOICES:
     1  Birthday Spacings
     2  Overlapping Permutations
     3  Ranks of 31x31 and 32x32 matrices
     4  Ranks of 6x8 Matrices
     5  Monkey Tests on 20-bit Words
     6  Monkey Tests OPSO,OQSO,DNA
     7  Count the 1`s in a Stream of Bytes
     8  Count the 1`s in Specific Bytes
     9  Parking Lot Test
    10  Minimum Distance Test
    11  Random Spheres Test
    12  The Sqeeze Test
    13  Overlapping Sums Test
    14  Runs Test
    15  The Craps Test
 Enter your choices, 1`s yes, 0`s no. using 15 columns:
123456789012345
000000011010110
  Starting time: 22:41:36    ::::::::::::::::::::::::::::::::::::::::::::::::::
::::::::::::::::
    ::     This is the COUNT-THE-1's TEST for specific bytes.          ::
    :: Consider the file under test as a stream of 32-bit integers.   ::
    :: From each integer, a specific byte is chosen , say the left-   ::
    :: most::  bits 1 to 8. Each byte can contain from 0 to 8 1's,    ::
    :: with probabilitie 1,8,28,56,70,56,28,8,1 over 256.  Now let    ::
    :: the specified bytes from successive integers provide a string ::
    :: of (overlapping) 5-letter words, each "letter" taking values  ::
    :: A,B,C,D,E. The letters are determined  by the number of 1's,  ::
    :: in that byte::  0,1,or 2 ---> A, 3 ---> B, 4 ---> C, 5 ---> D,::
    :: and  6,7 or 8 ---> E.  Thus we have a monkey at a typewriter   ::
    :: hitting five keys with with various probabilities::  37,56,70,::
    :: 56,37 over 256. There are 5^5 possible 5-letter words, and     ::
    :: from a string of 256,000 (overlapping) 5-letter words, counts ::
    :: are made on the frequencies for each word. The quadratic form ::
    :: in the weak inverse of the covariance matrix of the cell       ::
    :: counts provides a chisquare test::  Q5-Q4, the difference of  ::
    :: the naive Pearson  sums of (OBS-EXP)^2/EXP on counts for 5-    ::
    :: and 4-letter cell counts.                                      ::
    ::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
  Test results for erc41.txt
Chi-square with 5^5-5^4=2500 d.of f. for sample size: 256000
                      chisquare  equiv normal  p value
```

5- The result of test

```
::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
::                  THIS IS A PARKING LOT TEST                            ::
:: In a square of side 100, randomly "park" a car---a circle of           ::
:: radius 1.   Then try to park a 2nd, a 3rd, and so on, each             ::
:: time parking "by ear".  That is, if an attempt to park a car           ::
:: causes a crash with one already parked, try again at a new             ::
:: random location. (To avoid path problems, consider parking             ::
:: helicopters rather than cars.)   Each attempt leads to either          ::
:: a crash or a success, the latter followed by an increment to           ::
:: the list of cars already parked. If we plot n:  the number of          ::
:: attempts, versus k::  the number successfully parked, we get a         ::
:: curve that should be similar to those provided by a perfect            ::
:: random number generator.  Theory for the behavior of such a            ::
:: random curve seems beyond reach, and as graphics displays are          ::
:: not available for this battery of tests, a simple characteriz         ::
:: ation of the random experiment is used: k, the number of cars          ::
:: successfully parked after n=12,000 attempts. Simulation shows          ::
:: that k should average 3523 with sigma 21.9 and is very close           ::
:: to normally distributed.  Thus (k-3523)/21.9 should be a st-           ::
:: andard normal variable, which, converted to a uniform varia-           ::
:: ble, provides input to a KSTEST based on a sample of 10.               ::
::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
      CDPARK: result of ten tests on file erc41.txt
       Of 12,000 tries, the average no. of successes
            should be 3523 with sigma=21.9
       Successes:    1     z-score:******** p-value: .000000
       Successes:    1     z-score:******** p-value: .000000
       Successes:    1     z-score:******** p-value: .000000
       Successes:    1     z-score:******** p-value: .000000
       Successes:    1     z-score:******** p-value: .000000
       Successes:    1     z-score:******** p-value: .000000
       Successes:    1     z-score:******** p-value: .000000
       Successes:    1     z-score:******** p-value: .000000
       Successes:    1     z-score:******** p-value: .000000
       Successes:    1     z-score:******** p-value: .000000
      square size   avg. no.  parked   sample sigma
        100.            1.000            .000
       KSTEST for the above 10: p= 1.000000

$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
::                 THE 3DSPHERES TEST                                     ::
:: Choose  4000 random points in a cube of edge 1000.  At each            ::
```