

Image Encryption Systems

Innovation Lab CS299

Kavya Goyal 1901CS30

ABSTRACT

In this era of extensive sharing of media, security has been a major concern for many. Over time, many different architectures and algorithms were seen in order to secure or encrypt data like media files, text, documents etc. A multimedia data, an image for example, is a popular form of media exchange. Several image encryption frameworks have been developed and yet improving. Some of the parameters that should be kept in mind while making an image encryption algorithm are high key space and security and resistance against attacks. Unlike text files, multimedia information including image data has some special characteristics like high pixel redundancy and high correlation among pixels. This is the reason why text encryption algorithms like AES and DES fail. An encryption algorithm needs to undergo various attacks to be passed as a successful crypto system.

PROPOSED ALGORITHM

In order to implement an Image Encryption Algorithm, we need an image (also known as the plain image) and a secret key to convert our image into an encrypted image (also known as cipher image).

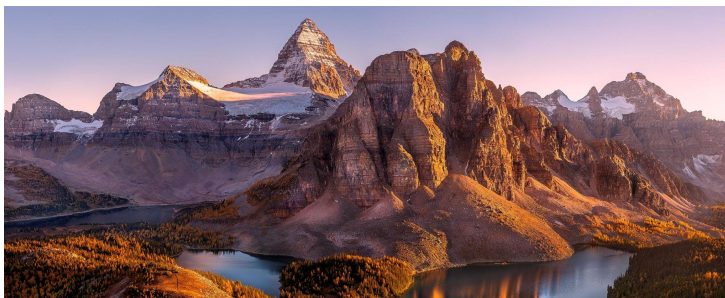


Fig 1. Mountain: 2383x952
(For further reference Elaine.jpg will be used for tests)



Fig 2. Elaine: 256x256

STEP 1: INITIALIZATION

1. Use a secret key of 156 bits
`0x12d6870025ad0e01d32ec40096b438074cbb1`
2. Scale the image to single channel
If the image is not grayscale, then convert it into a single channel image
3. Split the secret key in 5 parts (x_0, y_0, p_1, z_0, p_2) and run through the RandomNumber Generator Once to get Pseudo Random x_0, y_0, p_1, z_0, p_2 from the secret key

```

p2 = key & 0x7fffffff #31 bits
z0 = (key>>31) & 0x7fffffff #31 bits
p1 = (key>>62) & 0x7fffffff #31 bits
y0 = (key>>93) & 0x7fffffff #31 bits
x0 = (key>>124) & 0xffffffff #32 bits
x0,y0,z0 = InitPRNG(x0,y0,p1,z0,p2)

```

Fig 3. Splitting and Initialization of Inputs (seeds of Pseudo Random Number Generator (PRNG)) from key

STEP 2 : RANDOM NUMBER GENERATION

Two different kinds of chaotic maps are used as described below:

Piecewise Linear Chaotic Map (PWCM)

This is a uniform invariant distribution and very good ergodicity.

Piecewise linear chaotic maps are linear in parts over some functions

$$x_{n+1} = F(x_n, p) = \begin{cases} \frac{x_n}{p} & 0 \leq x_n < p \\ \frac{x_n - p}{0.5 - p} & p \leq x_n < 0.5 \\ F(1 - x_n, p) & 0.5 < x_n < 1 \end{cases}$$

Eq 1. Piecewise LCM

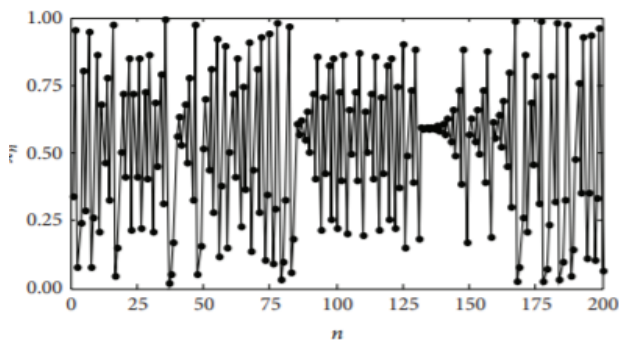


Fig 4. Random Numbers of PWLCM

Logistic Maps (LM)

Dependent on the simple equation of

$$x_{n+1} = H(x_n, \mu) = \mu \times x_n \times (1 - x_n)$$

Eq 2. Logistic Map

x_{n+1} values:

For $\mu < 3$: Remains constant/rises

For $\mu (3, 3.5)$: Oscillates between two values

For $\mu > 3.5$: Splits

At $\mu = 4$: We get sufficient number of observations

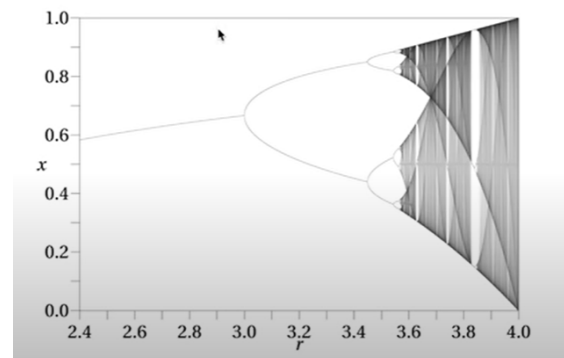


Fig 5. Bifurcation Diagram in Logistic Map

The proposed model uses a combination of Logistic Map and Piecewise Linear Chaotic Maps to generate 32 bits Pseudo Random Numbers which then make Permutation Matrix/Key and Diffusion Matrix.

$x_0 \rightarrow$ Fed as seed to Logistic Map \rightarrow Output after n iterations = x_n

$y_0, p_1 \rightarrow$ Fed as seed to $PWLCM_1 \rightarrow$ Output after n iterations = y_n, p_1

$z_0, p_2 \rightarrow$ Fed as seed to $PWLCM_2 \rightarrow$ Output after n iterations = z_n, p_2

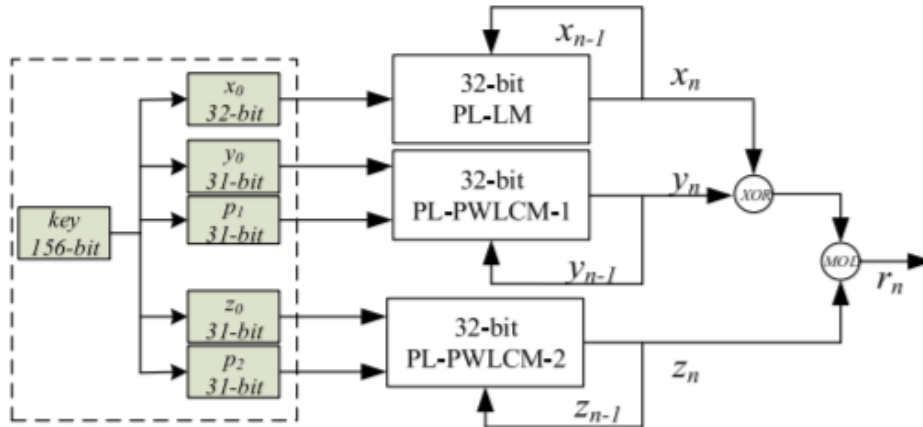


Fig 5. Random Number Generation by PWLCM + LM

The 32 bitPseudo Random Number R_n generated is given by

$$r_n = ((x_n \text{ XOR } y_n) + z_n) \text{ Mod } 2^L$$

Eq 3. Pseudo Random Number Generated

The x_n, y_n, z_n generated at each step is fed input at the next steps and act as (x_0, y_0, z_0) for the next round of pseudo random number generation.

STEP 3: PERMUTATION

Permutation or Confusion is the process by which the pixels of an image are scattered or replaced in order to remove the high redundancy of pixels. The permutation used in this algorithm is pixel level permutation using 4 units of pixel as an individual unit.

- For an image of size (w, h) , a list of pseudo random numbers is generated of size $(w \cdot h)/4$ from the combination of Logistic Map and Piecewise Linear Chaotic Maps
- The list is sorted according to the ascending order of the Random Numbers and their indexes are used for the permutation

				Random Number		Sorted Random Numbers	
Before Permutation				Index	Rn	Index	Rn
1	2	3	4	0	33	1	10
5	6	7	8	1	10	0	33
9	10	11	12	2	97	3	45
13	14	15	16	3	45	2	97
After Permutation							
3	4	1	2				
7	8	5	6				
11	12	9	10				
15	16	13	14				

Fig 6 Permutation Process

STEP 4 : DIFFUSION

After Permutation, we are left with the process of Diffusion. Diffusion is the process by which the units of an image are merged or diffused to change the values of individual pixels so as to decrease correlations between image pixels and increase entropy.

For the purpose of this algorithm, the unit of diffusion is chosen to be 1 row and 1 column.

- For a wxh image, the diffusion matrix required is of size WxH (**Where each number is an 8 bit number (between 0-255)**). This can be generated by splitting the 32 bit pseudo random number into 4 parts.
- Each row is diffused with the current row, the previous row and the corresponding row of the ValueKey (Diffusion Matrix). **The same process is done for the columns**

```

for iwhole in range(diffRound):
    for iCol in range(h-1,-1,-1):
        if iCol == 0:
            c[:, iCol] = (c[:, iCol] - c[:, -1] - ValueKey[:, iCol]) % scale
        else:
            c[:, iCol] = (c[:, iCol] - c[:, iCol-1] - ValueKey[:, iCol]) % scale
    for iRow in range(w-1,-1,-1):
        if iRow == 0:
            c[iRow, :] = (c[iRow, :] - c[-1, :] - ValueKey[iRow, :]) % scale
        else:
            c[iRow, :] = (c[iRow, :] - c[iRow-1, :] - ValueKey[iRow, :]) % scale

```

Fig 7. Diffusion Process

STEP 5 : ENCRYPTION

The process of Encryption requires combination of Permutation and Diffusion

Step 1: Permutation(image)

Step 2: Diffusion(image)

	Mountain.jpeg	Elaine.jpg
Variance of Original Image	4580.630705554325	2951.6775443117003
Variance of Encrypted Image	5457.256750016353	5474.248215612838
Average Local Entropy	7.902961096655099	7.902453308370529
Correlation Coefficient with original image		-0.01150943297947485
Encryption Time(s)	16.23983025550842 s	0.7389154434204102
Encrypted Image		

In order to exploit flaws and vulnerabilities in this process, many tests and analysis(s) were done

The permutation key is formed of pseudorandom numbers whose initial seeds are the parameters provided by the key.

```
Key1 = 0x000000000000000000001000000001000000001  
Key2 = 0x0000000000000000000010000000010000000010
```

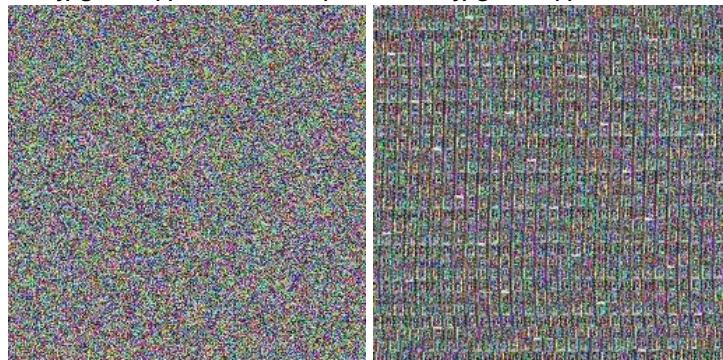
For a 256X256 GrayScale Image, the dimensions of the permutation matrix needed = $(256 \times 256) / 4 = 16384$. The parameters extracted in each case (After n=20 iterations)

	Key 1	Key 2
x0	0	0
y0	0	0
p1	1024	1024
z0	31457282	32
p2	1	16
Mean	288485633.45617676	32
Median	262146.0	32
Modes	514, 1026, 130, 258, 2050, 4098, 8194, 16386, 32770, 65538, 131074, 262146, 524290, 1048578, 2097154, 4194306, 8388610, 16777218, 33554434, 67108866, 134217730, 268435458, 536870914, 1073741826	32

Table 2. Initial Parameters Extracted from Key 1, Key 2

Where Mean, Median, Modes are for the set of Pseudo Random Numbers generated for the Permutation Key (Over 16384 observations)

Elaine.jpg Encrypted from Key 1 Elaine.jpg Decrypted from Key 2



Elaine.jpg Encrypted from Key 2 Elaine.jpg Decrypted from Key 1

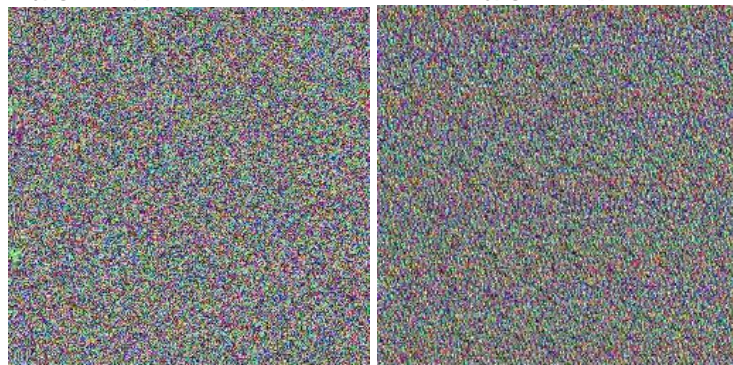


Fig 8. Decryption with wrong key

RESULTS

1. For the above keys, the x_0, y_0 components are zero implying that the Pseudo Random Number generated for the Permutation Matrix is dependent on the z_0 component fed as a seed in the PWLCM.
2. Even with a slight change, the image was not decrypted.
3. The PseudoRandom Numbers generated in each case were very different due to the difference in the initial seed p_2 .
4. It is better to have a key which has positive x_0, y_0 components.
5. The exact key is needed to get the desired permutation matrix.

2. Estimation of the Permutation Matrix without the Key

- One of the attacks is to estimate the permutation matrix without the key.

Let us encrypt an image using the Key 1

Key 1 = 0x0000000000000000000000001000000001000000001

Where $(x_0, y_0, p_1, z_0, p_2) = (0, 0, 1024, 31457282, 1)$ after $n = 20$ iterations

These are fed into the Random Number Generators as inputs to generate Pseudo Random Numbers

For a 256X256 GrayScale Image, the dimensions of the permutation matrix needed
 $= (256 \times 256) / 4 = 16384$

This is the number of Pseudo Random Numbers generated and needed for the permutation key. The following were found for these PseudoRandom Numbers

Mean	288485633.45617676
Median	262146.0
Modes	514, 1026, 130, 258, 2050, 4098, 8194, 16386, 32770, 65538, 131074, 262146, 524290, 1048578, 2097154, 4194306, 8388610, 16777218, 33554434, 67108866, 134217730, 268435458, 536870914, 1073741826

Table 3. Mean, Median Mode of Pseudo Random Numbers (16384 observations)

Now, instead of making a Permutation List consisting of Pseudo Random Numbers, we create a Permutation List consisting of constant numbers = R which would equal the Mean, Median or any one of the modes.

For the above key, the number of such Permutation Lists formed are = Mean(1) + Median(1) + Modes(24) = 26

Given by

```
PermutationKey 1 = listOf(Mean, Mean Mean, Mean, .....[16384 times])
```

```
PermutationKey 1 = listOf(Median,Median,Median,.....[16384 times])
PermutationKey 2 = listOf(Median,Median,Median,.....[16384 times])
```

PermutationKey 3 = listOf(Mode1,Mode1,Mode1[16384 times])

.....

```
PermutationKey 26 = listOf(Mode24,Mode24,Mode24 .....[16384 times])
```

On using these as PermutationKey for decrypting the encrypted image

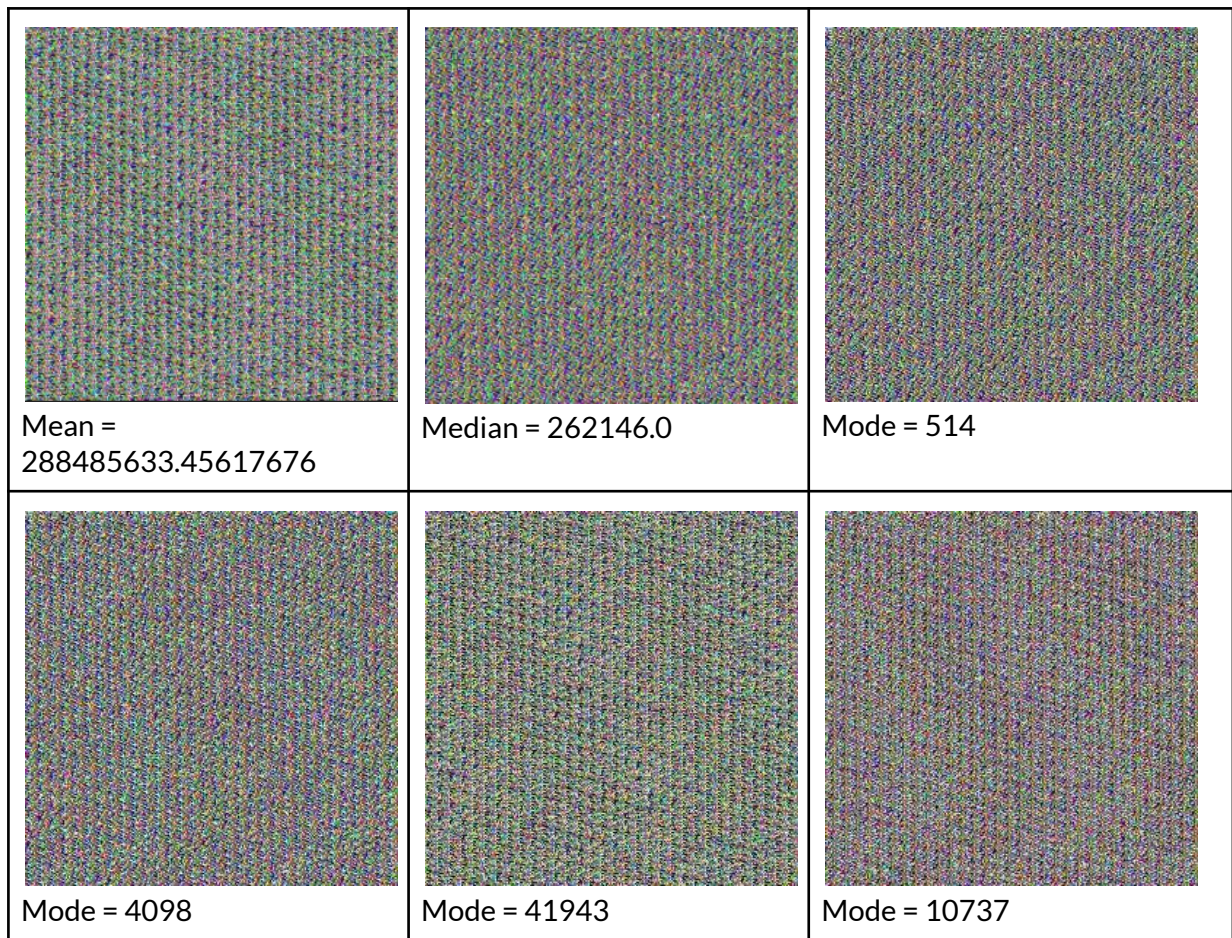


Fig 9. Image encrypted with Key 1 and decrypted with uniform permutation key

- An addition to the above test is to follow the same procedure done for R_n (PseudoRandom Numbers) on x_0, y_0, z_0 which act as the inputs.

Key 3 = 0x12d6870025ad0e01d32e74cbb1

$(x_0, y_0, p_1, z_0, p_2) = (0, 1871291175, 1511784598, 2714000237, 779406257)$

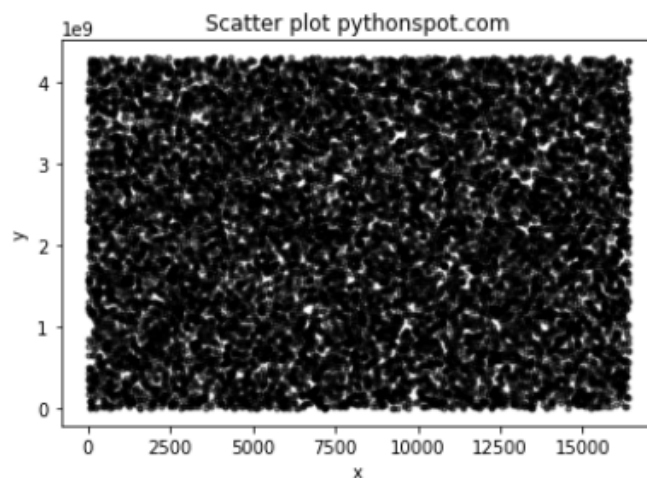


Fig 10. Scatter Plot for the Pseudo Random Numbers for Key 3 (16384 observations)

$x_{\text{Mean}} = 0$
 $y_{\text{Mean}} = 2152295453.350769$
 $z_{\text{Mean}} = 2136920512.8167114$
 $R_n = 4289215965 \text{ (11111111101010000011110111011101 = 32 bits)}$

The Permutation Key hence formed is
 $\text{PermutationKeyRn} = \text{listOf}(R_n, R_n, R_n, \dots [16384 \text{ times}])$

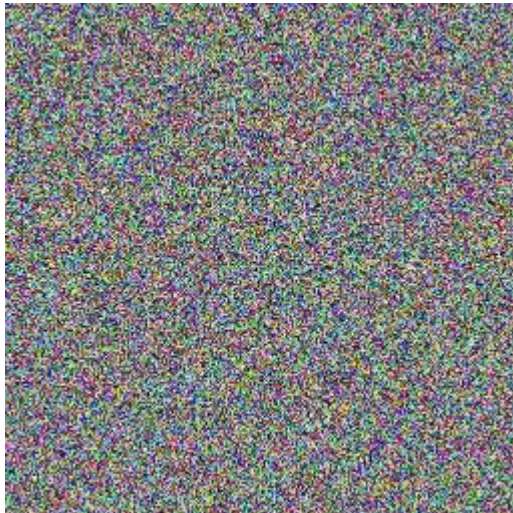


Fig 11. Elaine.jpg Encrypted from Key 3 and decrypted using Permutation Matrix R_n

(Note: We haven't used Key 1 here because $x_0, y_0 = 0$ and $x_n, y_n = 0$ for every n then after, so $R_n = (x^n y) + z = z$ always)

RESULTS

1. The Permutation Key created in this way is unable to decrypt the image

3. Analysis of the Key used in Encryption and security of PWLCM and LM

- Using a separate PseudoRandom Number Generator
 In order to randomize the numbers in the estimated Permutation Matrix, we have taken a simple 32-bit Pseudo Random Number Generator (Pure Logistic Map and Pure Piecewise Linear Chaotic Map separately) and look at the dependence of the Permutation Key on the Pseudo Random Number Generators and key parameters.

Let us encrypt and image with Key 1

Key 1 = 0x0000000000000000000000001000000001000000001

Where $(x_0, y_0, p_1, z_0, p_2) = (0, 0, 1024, 31457282, 1)$ after $n = 20$ iterations

We can see that the Random numbers generated are dependent on the x_0, y_0, z_0 as the following.

Logistic Map -> Numbers generated = 0 always x_0

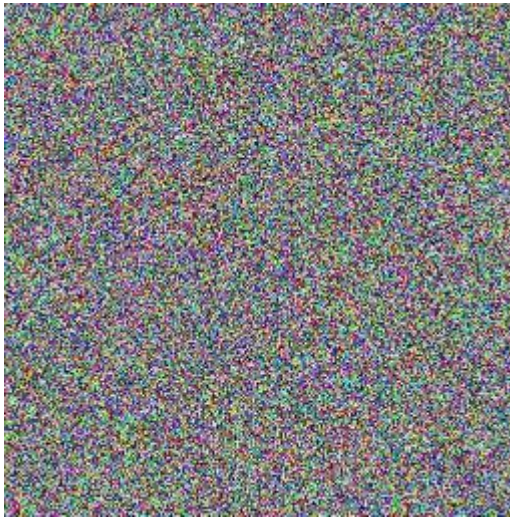
PieceWise Linear Chaotic Map (1) -> Numbers generated = 0 always y_0

The only effective component is the second PieceWise Linear Chaotic Map with z_0, p_2 as the input.

Therefore, after every n iterations to get the random numbers

$X_n = 0$, $Y_n = 0$, $Z_n = \text{Pseudo Random Number}$

- a. Let us consider a pure **Logistic Map** with the seed input $x_0 = 1234567$
Pseudo Random Numbers are generated using this x_0 in the Logistic Map which is then used to form a Permutation Key and decrypt the image encrypted with Key 1



Variances

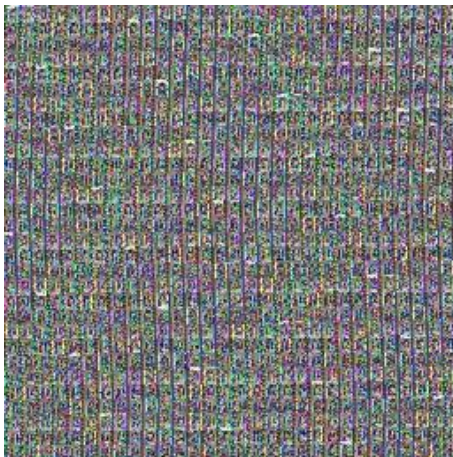
Original = 2951.6775443117003

Encrypted = 5478.615365683158

Decrypted = 5425.2161302733575

Fig 12. Image Encrypted with Key 1 and decrypted by creating a permutation matrix with Pseudo Random Numbers using a Random Number $x_0 = 1234567$ fed to pure Logistic Map

- b. Let us consider a pure **Logistic Map** with the seed input $x_0 = 0$
Pseudo Random Numbers are generated using this x_0 in the Logistic Map which is then used to form a Permutation Key and decrypt the image encrypted with Key 1



Variances

Original = 2951.6775443117003

Encrypted = 5478.615365683158

Decrypted = 3534.777441505217

Fig 13, Image Encrypted with Key 1 and and decrypted by creating a permutation matrix with Pseudo Random Numbers using a Random Number $x_0 = 0$ (**same as input**) fed to pure Logistic Map

Observations:

For an image, encrypted with a key with parameters $(x_0, y_0, p_1, z_0, p_2)$ as seeds for LM,

- If a Pure Logistic Map is used as a random number generator with seed $x_{\text{New}} \neq x_0$, then the image is not decrypted at all.
- If a Pure Logistic Map is used as a random number generator with seed $x_{\text{New}} = x_0$, then the image is decrypted to **some extent** and local entropy and variance decreases significantly as compared to the previous case

- c. Let us consider a pure **PWLCM** to be used as a chaotic map with different seeds (y,p)
Pseudo Random Numbers are generated using these y,p PWLCM which is then
used to form a Permutation Key and decrypt the image encrypted with Key 1

 <p>y=123456 p= 1024</p>	 <p>y=0 p =1024</p>	 <p>y=31457282 p= 1 (Original Parameter(Seed = y0))</p>
 <p>y=65538 p=1 y0 after n iterations</p>	 <p>y=131074 (Next number after seed yn) p=1</p>	 <p>y=65539 p=1</p>

Table 4. Results of using pure PieceWise Linear Chaotic Maps as Pseudo Random Numbers with variable inputs (seeds) y,p to decrypt an image encrypted with Key 1

RESULTS

1. The Encryption Key should be such that it is not easy to guess and is such that non-zero parameters are fed as seeds in the Logistic Map and Piecewise Linear Chaotic. This is vital so that the common pseudo random number generators like LM or PWLCM do not generate the Random Numbers when given zero parameters
2. The key should be at least 125 bits long ($156-31*4$) to assign a non zero value to all the parameters
3. It is crucial that the seeds p1,p2 are non-zero for the Algorithm to work because PWLCM part of Random Number Generation has division by p1,p2 type operations

Example: For the Secure Key

0x74cbb174cbb174cbb174cbb174cbb174cbb1

x0=306565294

y0=2453551650

p1=1395574227

z0=185973024

p2=829737905



Fig 14. Results of using Pure PWLCM as PRNG with secure key showing no information of the plain image even after using the exact parameters used in one PWLCM

4. Analysis of the Cipher Image to Plain Image

One of the fundamental properties that any image encryption algorithm holds is its resistance against attacks. While doing cryptanalysis of images, some of the features to keep in mind are

- Amount of time available
- Computing power available
- Storage capacity available

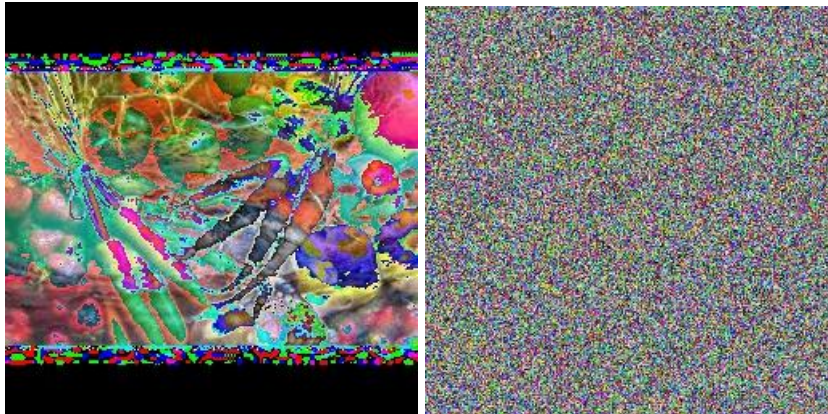
Brute Force Attack- Due to the large key space of 156 bits, it is not possible to try all the input combinations of the key. A large key is preferred to avoid brute force attacks.

Chosen Plaintext Attack (CPA) – In this attack, we have the image and its corresponding cipher image. An example of this attack is *differential cryptanalysis*.

In a broad sense, differential cryptanalysis is a cryptanalytic method studying how particular differences in plaintext pairs affect the resultant differences, which is also called differential of the corresponding cipher texts.

- ❖ A sample of 10 images was taken. All the images were reshaped to the same size and then encrypted with the same key. As they have the same key, they will have the same permutation matrix and diffusion matrix
- ❖ Key used = 0x74cbb174cbb174cbb174cbb174cbb174cbb1
- ❖ Differential Analysis on a set of 10 images encrypted with the same key. Number of combinations = $^{10}C_2 = 45$
- ❖ Out of the set of all the plain images, the difference between every two pairs of plain images was found.
- ❖ Also the corresponding difference between every two pairs of cipher images was found.

The objective of finding the correlation between the difference in these pairs of images and ciphers is to find out the effect that changing an input plain image has on the cipher image when encrypted with the same and also gaining information about the key in any form if possible



_org_4_7.jpg (Image 4 - image 7) _enc_4_7.jpg (Enc Image 4 - Enc image 7)

Fig 15. Differences in plain images and cipher images

Pearson's Correlation Coefficient (PCC) r was used to establish linear correlations between the differences of the pair of plain images and cipher images. It's often denoted with the letter r and called Pearson's r . It has been found that PCC overstates its dependence. So, r^2 is considered a more optimum relation

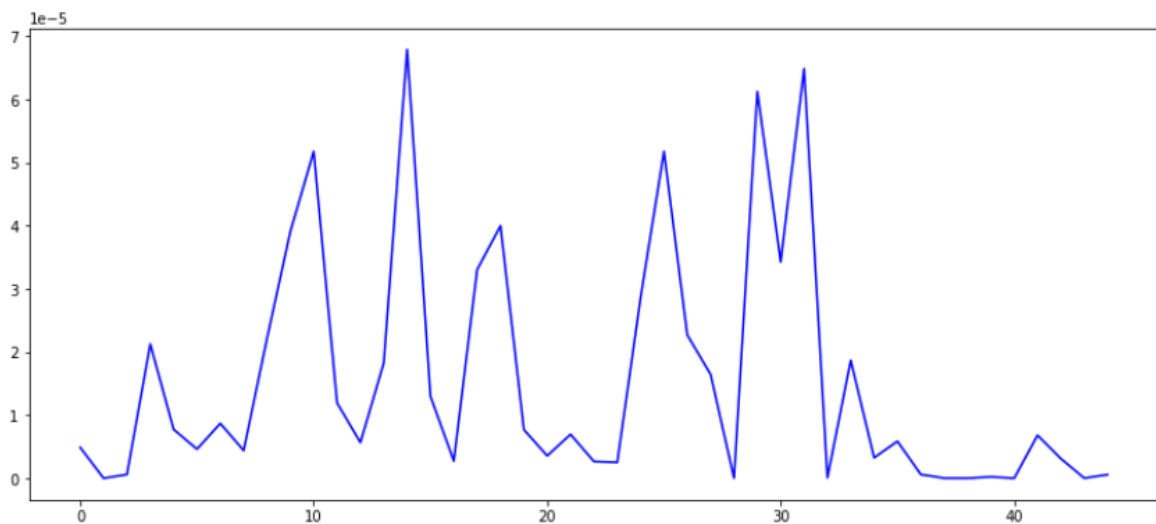


Fig 16. Linear Correlation Coefficient Graph r^2 vs index for the observation with Difference Relation

In order to get more insights on whether these linear Pearson correlations are significant or not, Fisher's z-Transform is used.

```

#r = Pearson Coefficient
z = math.atanh(r)
SD_z = 1 / math.sqrt(w*h - 3)
z_upper = z + 1.96 * SD_z
z_lower = z - 1.96 * SD_z
r_upper = math.tanh(z_upper)
r_lower = math.tanh(z_lower)

#If r_upper and r_lower lie on the same side of zero, there there exists a statistical significance
if r_upper*r_lower > 0:
    print('Statistical Significance found for image pairs ', i,',',j)
    return 1
else:
    return 0

```

Fig 17. Fisher's z Transform

For the taken sample of 45 image pairs, the statistical significance of Fisher's Transform of their Pearson Linear Correlation Coefficients is given by

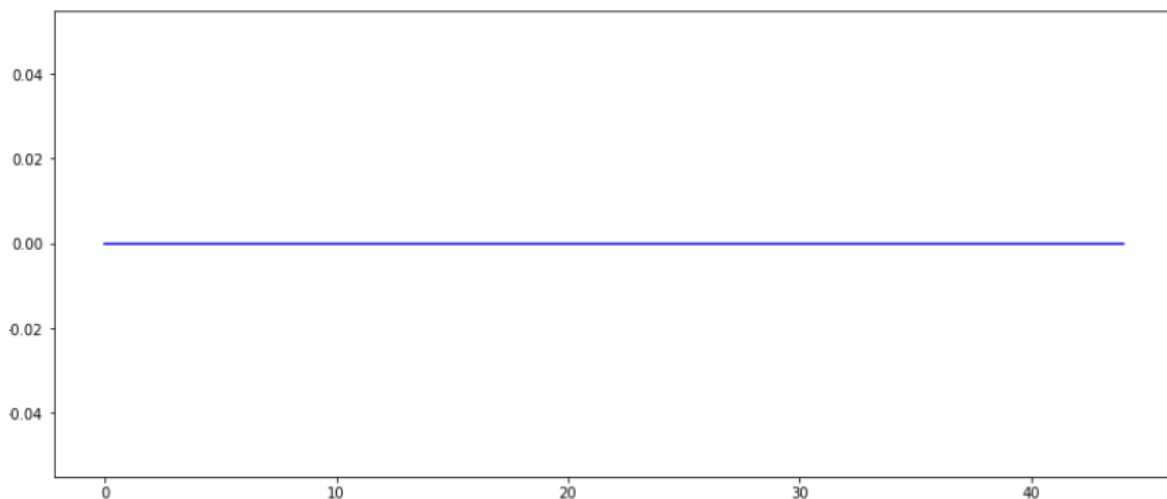


Fig 18. Statistical Significance (0= No relation, 1 Relation present)

Observations:

- For a sample of 10 images of the same size encrypted using the same key, the correlation between every pair of images and their corresponding encrypted image has a linear correlation varying between (0,0.75). **That is, the difference in two plain images matches/correlates the difference in their cipher images by at max 0.75%.**
- After using Fisher's Transform, it was found that though there exists a positive linear correlation between the differences in plain image and cipher image, it is negligible and is not statistically significant

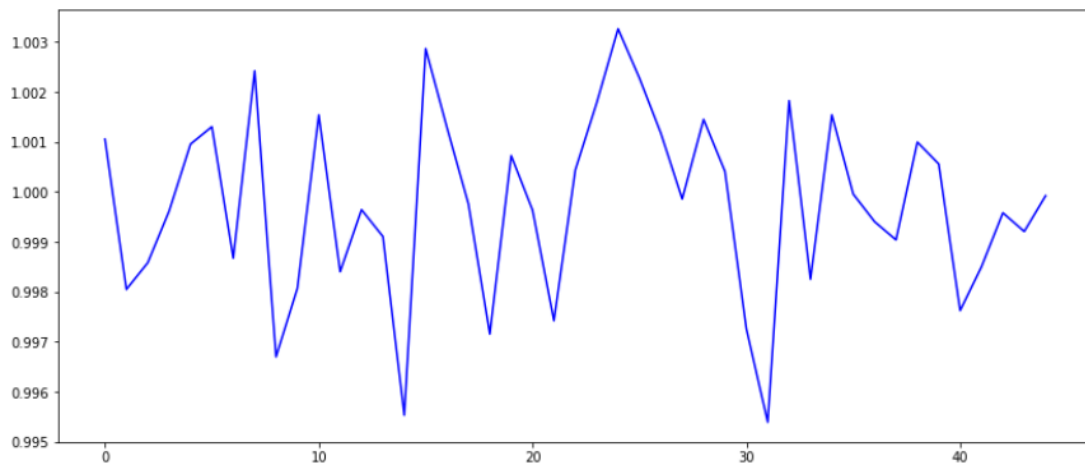


Fig 19. Distance Relation between the image and cipher difference images

The distance mentioned by Fig 19 has a range of $[0,2]$ where

0 : Perfectly positively correlated

1: No correlation

2: Perfectly negatively correlated (Anti Correlation)

RESULTS

1. The Linear Correlations found out by Pearson Correlation Coefficient which are significantly low show that, on two different images, the cipher images encrypted using the same keys show little to no linear correlation.
2. Same is true for Non-Linear relationships of these image-cipher pairs.
3. The effects of changing the plain image are not distinguishable or related to the change in their cipher image.
4. No information about the key being used was found

Ciphertext Only Attacks (COA) – In this attack, we have access to a set of cipher images. That is, the same image encrypted with different keys. The following test is to check for vulnerability against this test

Let us consider an image of size 256x256, it was encrypted with 14 different keys

0x74cbb174cbb174cbb174cbb174cbb174cbb174cbb1.

0x74cbb174cbb174cbb174cbb174cbb174cbb174cbb2,

•

0x12d6870025ad0e01d32ec40096b438074cbb4.

```
0x0000000000000000000000001000000001000000001.
```

```
0x00000000000000000000000000000000100000000010000000010
```

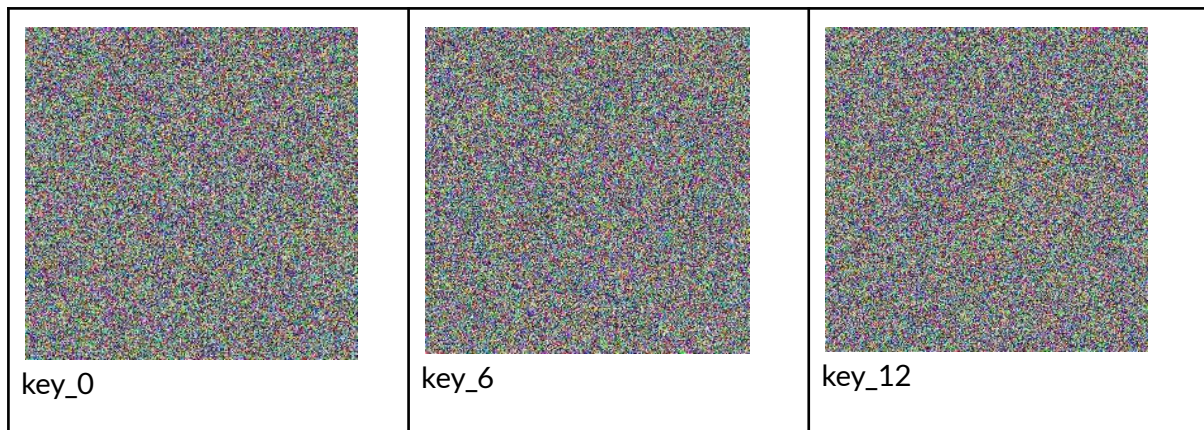


Fig 20. Image encrypted with different keys

For 14 keys ,the number of ciphers made = 14.
 Number of pairs for analysis = $^{14}C_2 = 91$

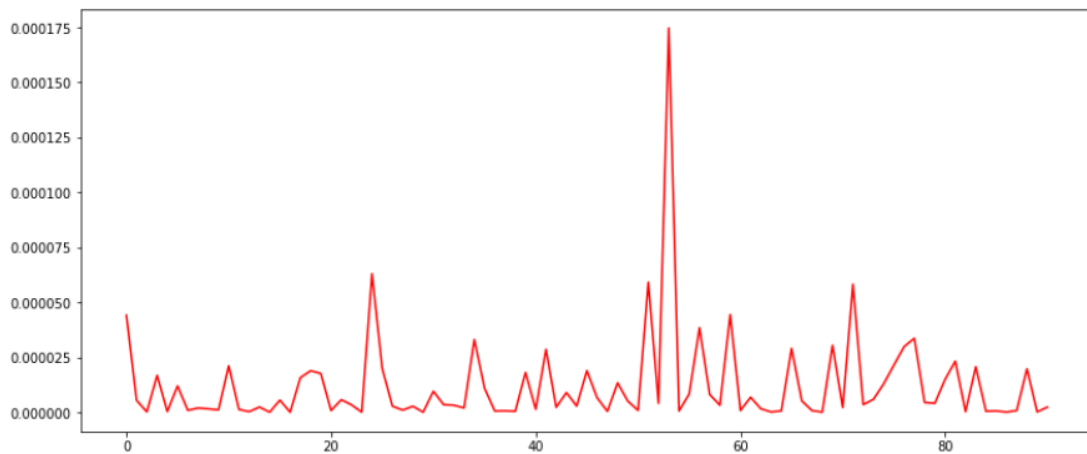


Fig 21. R^2 Linear correlations between all pairs of ciphers

Linear Correlations were found between all the ciphers to see for any relation between them and results are shown in Fig 21.

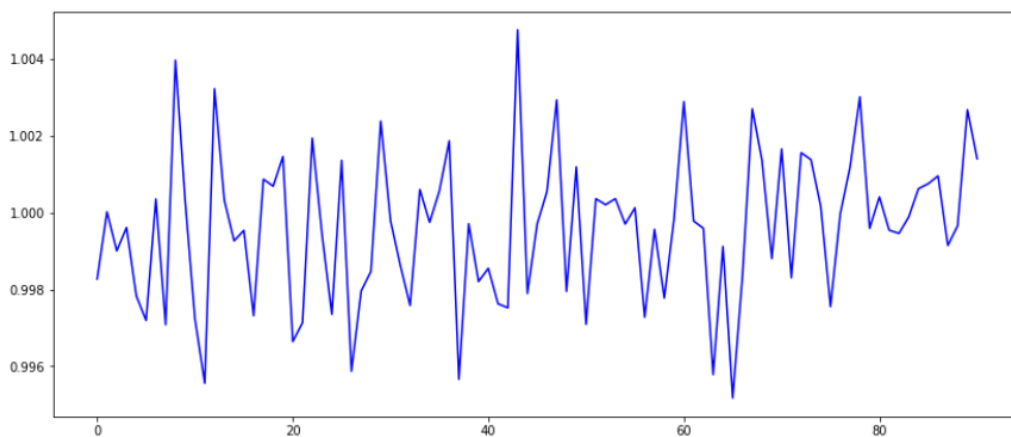


Fig 22. Distance Correlations between xor of all pairs of ciphers with the original image

Also, for all cipher images C_i and C_j ($i \neq j$), the relation/dependence

$C_i \text{ xor } C_j$ with the original image was found (Shown in Fig 22)

The distance correlation being close to 1 signifies negligible dependence between the original image and XOR of a pair of cipher

Local Entropy of original Image = **6.181840342915623**

We then obtain the Local Entropy of xor of every pair of ciphers with the encrypted image. (Shown in Fig 23)

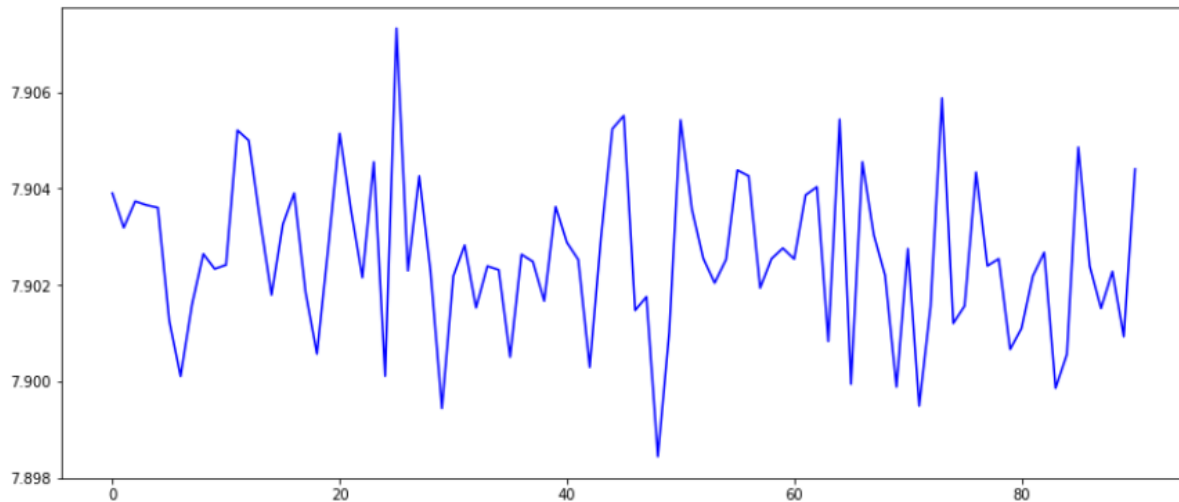


Fig 23. Local Entropy between xor of all pairs of ciphers with the original image

RESULTS

1. From Fig 21 and Non-Linear relations between the cipher pairs show that they have negligible relationship despite being encrypted from the same plain image
2. Every pair of cipher which is encrypted from the same plain image, (XOR)ed together have no appreciable relation with the original image
3. The entropy of every pair of cipher (XOR)ed together is quite high as compared to the original image proving that the ciphers give out no information of the plain text

6. Reversing the Algorithm and obtaining a temporary diffusion matrix

For the decryption process, the first step is to de-diffuse in reverse order. (Starting from the end from columns and then moving to rows, Fig 7)

In order to gain an insight on what could happen if we would reverse these processes with taking the cipher and the plain image, the following transformations were used to obtain the value key or the Diffusion Matrix

```

for j in range(h-1,-1,-1):
    if j == 0:
        value[:,j] = (imgEnc[:,j] - imgOrg[:,j]-imgEnc[:,-1])%scale
    else:
        value[:,j] = (imgEnc[:,j] - imgOrg[:,j]-imgEnc[:,j-1])%scale
for j in range(w-1,-1,-1):
    if j == 0:
        value[j,:] = (imgEnc[j,:] - imgOrg[j,:]-imgEnc[-1,:])%scale
    else:
        value[j,:] = (imgEnc[j,:] - imgOrg[j,:]-imgEnc[j-1,:])%scale

```

Fig 24. Snippet for Reverse Code Diffusion
(Where diffRound = 2 by convention and ValueKey is the diffusion matrix)

On using the **value[]** thus obtained as a Diffusion Matrix and producing the permutation key from it to decrypt the image, the following result was found

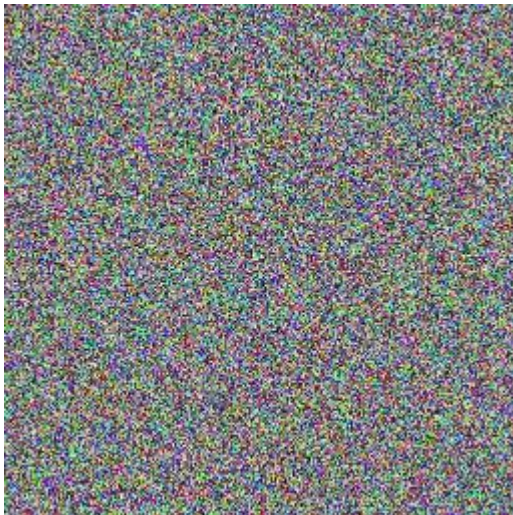


Fig 25. Decrypted Image from **valueKey**

RESULTS

1. The diffusion was dependent on the current column of the previous state and previous row of the current state where the previous state was not original image for current plain image, therefore we were not able to get any information of the encrypted image

VULNERABILITIES FOUND

ISSUES	SOLUTIONS
Calculation complexities with recursions, calculations around floating point numbers generated by chaotic Map and problems with infinite precision	Rounded off to the greatest integer function and using 32 bit precision for Pseudo Random Number Generation
Insecure Keys Leading to loss of inputs fed to PWLCM_LM making it prone to obtaining key and decrypting by Single Random Number Generators (3)	Keys of Minimum size 125 and Preferably 156
Invalid Key For a 156 bit key, the first step is to divide it into 5 parts (x_0, y_0, p_1, z_0, p_2). After generation of pseudo random numbers, x_0, y_0, z_0 changes to x_n, y_n, z_n however p_1, p_2 remain the same. In the generation, p_1, p_2 numbers as can be seen in Eq Therefore p_1, p_2 can never be zero	Validity of Key used
Permutation Unit of 4 pixels 4 pixels have been taken as the unit of permutation in order to reduce time complexity. The close dependence of each of these pixels can be exploited as well to improve randomness	Using a different Permutation Scheme

Table 5. Vulnerabilities found

AUGMENTATION ON THE MODEL

After going through various techniques and tests to overcome the numerous vulnerabilities exposed by the model, I hereby propose a number of small augmentations in the proposed algorithm to improvise it and protect it against its vulnerabilities.

1. To add a check on the key used so that it is greater than 125 bits at least
To protect the algorithm against weak and invalid keys, it is necessary that the key is at least 125 bits long and the input seeds are not 0.

```
def validateKey(key):

    try:
        x0,y0,p1,z0,p2 = getInitialSeeds(key)

        if x0 == 0 or y0 == 0 or z0 == 0 or p1 == 0 or p2 == 0:
            print('The key is not secure')
            return False

        else:
            print('The key is safe to use')
            return True

    except (RuntimeError, TypeError, NameError):
        #Case when p1 or p2 extracted 0 (Runtime Error)
        print('The key is not valid')

        return False
```

Fig 26. Validation and Security of the Key Used

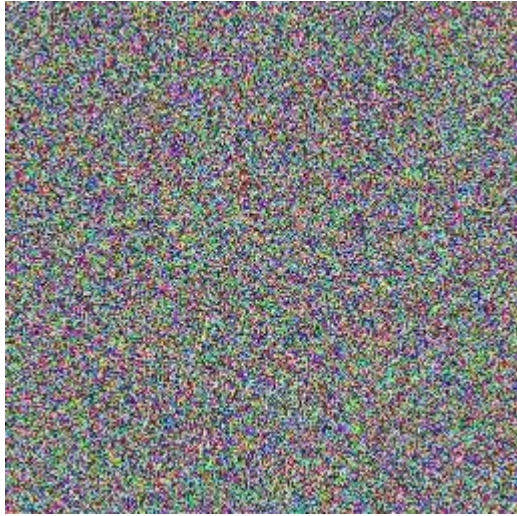
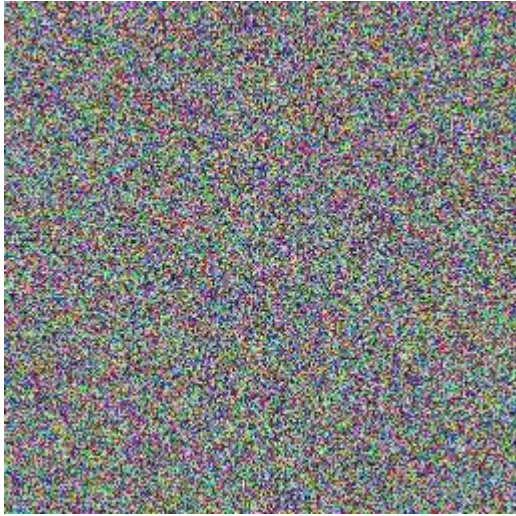


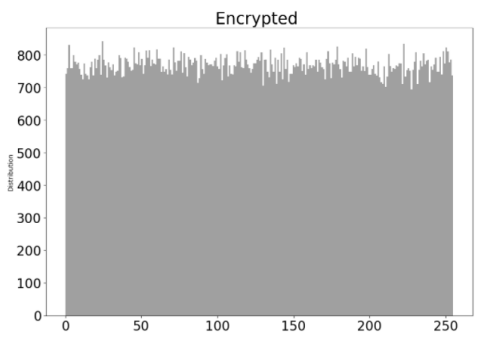
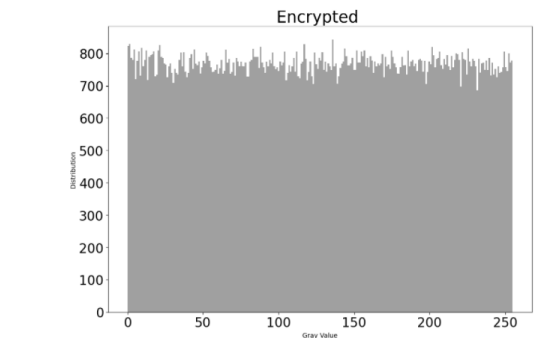
2. Using a Pixel Level Permutation Combined with Row-Col Permutation to remove the interdependence within one pixel unit as well

				Random Number		Sorted Random Numbers	
Before Permutation				Index	Rn	Index	Rn
1	2	3	4	0	33	1	10
5	6	7	8	1	10	0	33
9	10	11	12	2	97	3	45
13	14	15	16	3	45	2	97
After Pixel Permutation				Row RandomNumber		Using same for row, col	
				Index		Sorted Index	
3	4	1	2	0	33	0	
7	8	5	6	1	293	3	
11	12	9	10	2	43	1	
15	16	13	14	3	224	2	
Row							
3	4	1	2				
15	16	13	14				
7	8	5	6				
11	12	9	10				
Col							
3	2	4	1				
15	14	16	13				
7	6	8	5				
11	10	12	9				

Fig 27. Proposed Permutation Algorithm

This will take **(w+h)** extra time to permute the row and column

Table 6. Comparisons between the algorithms

	Augmented Algorithm	Original Model
Encrypted Image		
Decrypted Image (Elaine.jpg)		
Histogram		
Variance (Encrypted)	5474.503588851024	5463.266291051
Average Local Entropy (Encrypted)	7.90396778881896	7.902619692664385
Linear Correlation Coefficient with the original image	0.000989925796320189	-0.002555126682647107

NonLinear Distance Correlation with original image	0.9988728269431955	0.9972582663799644
Time for Encryption	1.038682222366333 s	0.7291602611541748 s

Increase in local entropy, values of Linear Dependence closer to 0 and Non-Linear distance correlations closer to 1 as compared to the original algorithm reflect improvement in the algorithm.

CONCLUSION

Image Encryption Algorithms are being constantly improved and tested against various attacks. Under this project, I learned about various frameworks and techniques used in Image Encryption and developed over years. I read about symmetric key encryption systems and chaotic maps and became familiar with how chaos theory and randomness are used in image encryption algorithms. I learned about various chaotic maps like Arnold Cat Maps, Henon Map, PieceWise Linear Chaotic Map and Logistic Maps. I came to know about various parameters on which the security of the image can be measured like entropy and correlations. I studied about various permutation and diffusion techniques and their complexities in terms of time. I worked closely on the algorithm of a paper which worked extensively to increase the security of the Pseudo Random Numbers generated by using a combination of PieceWise Linear Chaotic Map and Logistic Map. On analysing the keys used for the generation of the Pseudo Random Numbers, I came across certain vulnerabilities that needed attention for resistance against attacks.

I have proposed changes to a novel image encryption algorithm^[2] by taking into consideration the security of the key used and also an augmentation on the permutation algorithm to increase the entropy and randomness of the cipher image produced.

RESOURCES AND REFERENCES

1. [Kavya-24/Innovation-Lab-Project: CS299 Project](#)
2. [A Robust Image Encryption Algorithm Based on a 32-bit Chaotic System](#)
3. [Image Encryption Techniques Comparisons](#)
4. [Fast image encryption algorithm based on parallel computing system](#)
5. [Secure Image Encryption Algorithms: A Review](#)
6. [Piecewise Linear Chaotic Map](#)
7. [Introducing Distance Correlation, a Superior Correlation Metric. | by Terence Shin](#)