Image Cryptography

A Project Submitted
in Partial Fulfillment of the Requirements
for the Degree of

Bachelor of Technology
in
Computer Science and Engineering
by

Abhinav Kumar (Roll No. 34900119042)

Purnendu Roy (Roll No. 34900119027)

Rushali Sarkar (Roll No. 34900119022)

to

Department of Computer Science and Engineering
Coochbehar Government Engineering College
Coochbehar - 736170
May 2023

DECLARATION

We, **Abhinav Kumar (Roll No: 34900119042)**, **Purnendu Roy (Roll No: 34900119027)**, and **Rushali Sarkar (Roll No: 34900119022)**, hereby declare that this project entitled "**Image Cryptography**" submitted to Coochbehar Government Engineering College towards the partial requirement of **Bachelor of Technology** in **Computer Science and Engineering**, is an original work carried out by us, under the supervision of **Professor Sukhendu Shekhar Mondal**. We have sincerely tried to uphold academic ethics and honesty. Whenever a piece of external information or statement or result is used then, that has been duly acknowledged and cited.

Coochbehar - 736170
May 2023

Abhinav Kumar
Purnendu Roy
Rushali Sarkar

ACKNOWLEDGEMENT

We wish to extend our sincere and heartfelt obligation towards everyone who helped us see this project through till the end. We express our profound gratitude and deep regard to **Professor Sukhendu Shekhar Mondal**, **Dr. Sudip Kumar Adhikari**, and **Dr. Prasenjit Dey** for their guidance, valuable feedback, and constant encouragement throughout the project. We would like to sincerely acknowledge their valuable suggestions and constant support every step of the way.

We are eternally grateful to **Dr. Sourav De**, **Head of the Department** of **Computer Science and Engineering**, Coochbehar Government Engineering College for his steadfast encouragement, valuable support, advice and time. We are also grateful to **Coochbehar Government Engineering College** for providing the necessary resources and facilities to complete this project to the best of our ability.

Coochbehar - 736170
May 2023

Abhinav Kumar
Purnendu Roy
Rushali Sarkar

ABSTRACT

Name of Student: **Abhinav Kumar** Roll No: **34900119042**
Name of Student: **Purnendu Roy** Roll No: **34900119027**
Name of Student: **Rushali Sarkar** Roll No: **34900119022**
Degree for which Submitted**: Bachelor of Technology**
Department: **Computer Science and Engineering**
Project Supervisor: **Professor Sukhendu Shekhar Mondal**
Date of Project Submission: **24th May, 2023**

In this world of huge data transfers and specifically multimedia transformations a constant improvement of cryptographic algorithms is one of the major necessities for ensuring security and efficiency. We have made an effort to apply an image cryptographic algorithm using the concepts of KAA Map and a variety of chaotic maps including 2D Logistic Sine Map, Linear Congruential Generator, Tent Map and Bernoulli Map. The algorithm takes into account Shannon's idea of security such that the encryption is hugely carried out using bit confusion and bit diffusion. Confusion is created using 2 keys generated through the different chaotic maps and Diffusion is created using KAA Map. The encryption algorithm is also shown to successfully pass all the security analysis methods.

Index Terms: KAA Map, 2D Logistic Sine Map, Linear Congruential Generator, Tent Map, Bernoulli Map, Shannon's Theory

CONTENTS

INTRODUCTION

With the advancement of technology and security threats on data transfer, maintaining confidentiality of information has become more and more difficult. To tackle this hurdle, we have come across an image encryption algorithm that has different layers of encryption. Encrypting an image is more tricky than normal text due to the fact that adjacent pixels in an image have high redundancy and correlation.

We have attempted to keep in mind Shannon's idea of security while implementing the algorithm which states that high level of information security can be obtained by two mutually independent encryption stages, namely confusion and diffusion. A confusion stage forces every bit in an encrypted image to depend on many parts of the key, thus hiding the connection between the two. While a diffusion state introduces an avalanche effect, a change of a single bit in the plain image would result in change of roughly half of all the bits in the encrypted image. The purpose of diffusion is to eliminate any statistical relationship from being exhibited between the original image and the corresponding encrypted image.

It is in the design of these two stages of encryption where chaotic functions come into play. Chaotic functions exhibit a number of inherent characteristics that make their use advantageous in relation to communication security. Those characteristics include sensitivity to initial conditions, ergodicity, pseudo-randomness,

control parameters and periodicity, to name a few. In general, chaotic functions are classified either into one-dimensional (1D) or multi-dimensional (MD). The choice of adopting 1D over MD chaotic functions for their utilization in image encryption algorithms is always a matter of trade-off between complexity and security. One-dimensional chaotic functions provide simple software and hardware implementations at the price of acceptable security. This makes them ideal for image encryption applications requiring real-time efficiencies. On the other hand, MD chaotic functions provide excellent security but do achieve at the price of more complex designs and implementations.

The main algorithm used in this paper is as follows:

1) A color image encryption algorithm is proposed, utilizing 2 keys and a combination of 1D and MD chaotic functions, as well as the KAA map.
2) In order to reach a heightened level of security, the proposed algorithm utilizes both confusion and diffusion, satisfying Shannon's theories.
3) The proposed algorithm is tested against a number of visual, statistical and differential attacks, in order to measure its performance in terms of efficiency, robustness and resistivity to cryptanalysis.
4) By employing 2 keys and 9 variables, the key space can be enlarged to 2^{478}, resisting brute-force attacks.

CONCEPTS

- KAA MAP
  - GALLILEAN TRANSFORMATION
  - ROTATIONAL TRANSFORMATION
- 2D-LOGISTIC-ADJUSTED-SINE MAP
  - LOGISTIC MAP
  - SINE MAP
- LINEAR CONGRUENTIAL GENERATOR
- BERNOULLI MAP
- TENT MAP

GALILEAN TRANSFORMATION

Symmetrical transformations in physics are expected to keep physical quantities invariant. Each transformation keeps one physical quantity conserved or invariant.

We have observed that Galilean transformation is a linear transformation to be one such symmetrical transformation along one dimension.

It can be written as,

$$\mathbf{r}' = \mathbf{r} + \mathbf{v}t$$
$$t' = t$$

where r is the position of a particle in an inertial frame (x, y) and r' is the transformed position of the particle in the new inertial frame (x', y'), v is a constant in which $|v|$ is $<<$ c, is the linear velocity vector of the new frame and finally t and t' are invariant time variables.

The Galilean transformation keeps some physical quantities invariant. Among them, is the distance $\Delta r' = r_2' - r_1' = \Delta r = r_2 - r_1$.

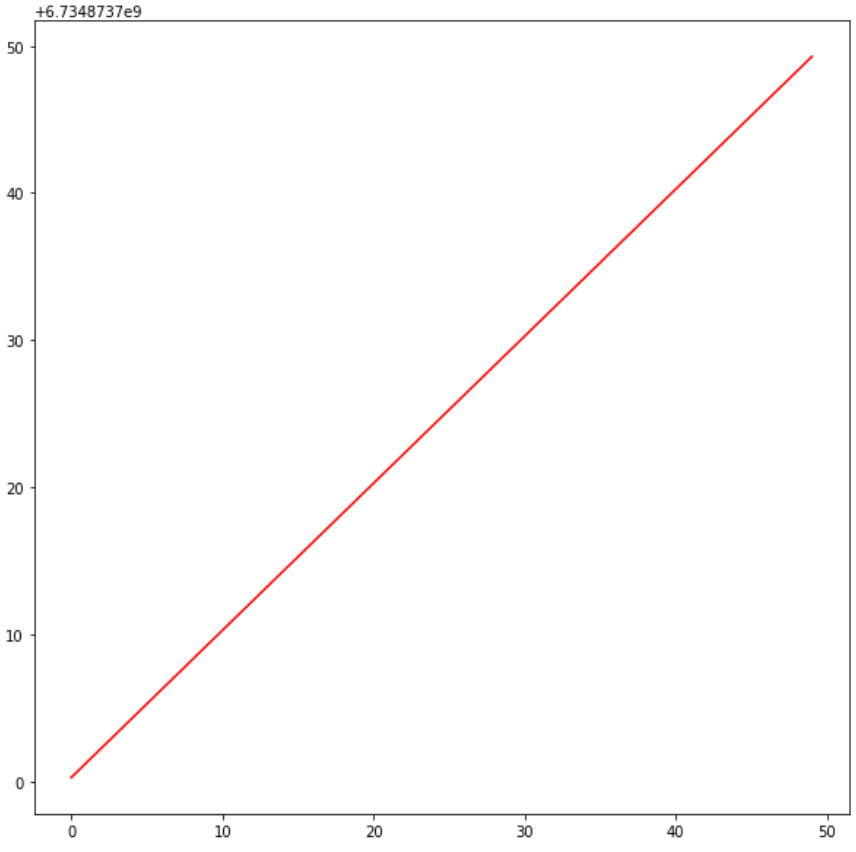
This can be demonstrated by,

In [2]:
```
def get_galilean_transformation(r, v, t):
    return r + v * t
```

In [3]:
```
x = [each for each in range(50)]
y = [get_galilean_transformation(each, 4, time.time()) for each in range(50)]
```

In [4]:
```
plt.figure(figsize = (10,10))
plt.plot(x, y, color="red")
```

Out[4]: [<matplotlib.lines.Line2D at 0x7fce494333d0>]



ROTATIONAL TRANSFORMATION

Second, the Rotational Transformation, which is defined as,

$$r' = Rr = \begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix} r,$$

$$t' = t$$

where θ is a constant rotation angle around the z–axis, for the frame r = (x, y) to rotate into the frame r' = (x', y').

Rotational symmetry transformation also keeps the distance Δr invariant in the new frame. It keeps the position distribution of points unchanged as well.

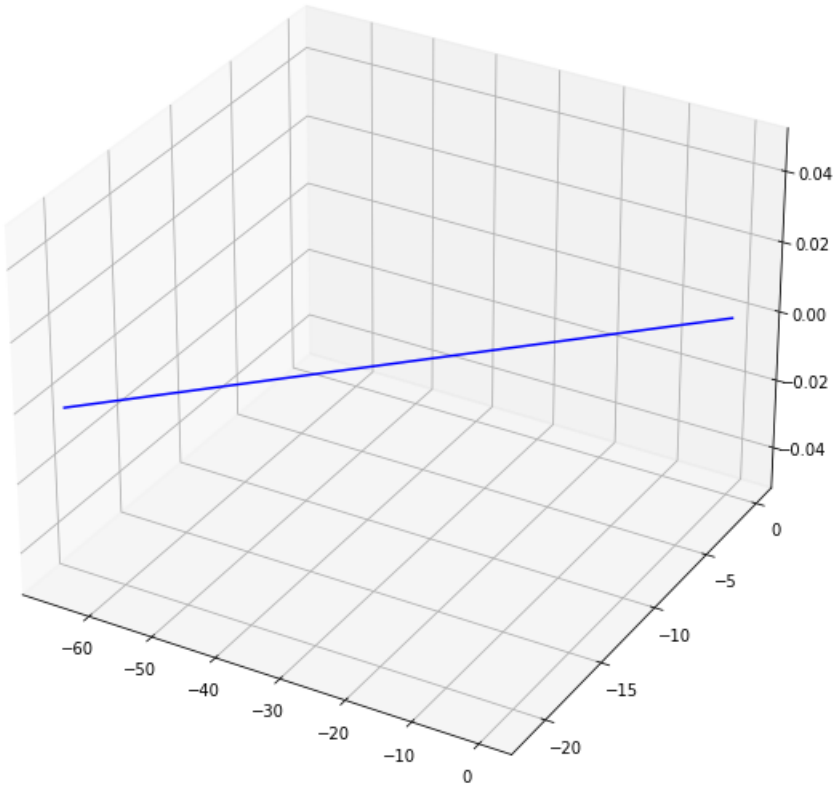
This can be demonstrated by,

```
In [5]: def get_rotational_transformation(x, y, theta):
            return x * math.cos(theta) - y * math.sin(theta), x * math.sin(theta) + y * math.cos(theta)
```

```
In [6]: points = [get_rotational_transformation(each, each, 2.67) for each in range(50)]
        x0, y0 = zip(*points)
```

```
In [7]: plt.figure(figsize = (10,10))
        plt.axes(projection='3d')
        plt.plot(x0, y0, color="blue")
```

Out[7]: [<mpl_toolkits.mplot3d.art3d.Line3D at 0x7f410e1917c0>]



KAA MAP

Therefore, the combined symmetry transformation, which is

$$r' = Rr + vt$$
$$t' = t$$

keeps the distance Δr invariant in the new frame and the distribution of the positions of particles (in this case, pixels) is maintained.

In the KAA map model, we introduce the transformation:

$$r' = R(\theta(t))r + v(t),$$

where the rotation angle θ is made time (step) dependent $\theta(t)$ and the linear velocity v is no longer constant but made time (step) dependent v(t), as well. The θ and v are polynomial functions that depend on the parameter t. This transformation destroys completely any symmetry of the group of positions *r* in the initial frame after being transformed into the new frame.

We introduce the KAA map finally as,

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \left[\begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} V_1 \\ V_2 \end{bmatrix} \text{ X t} \right] mod\ N,$$

where mod N is used to confine the numbers from 0 to 255 which is the minimum and maximum value a pixel can attain.

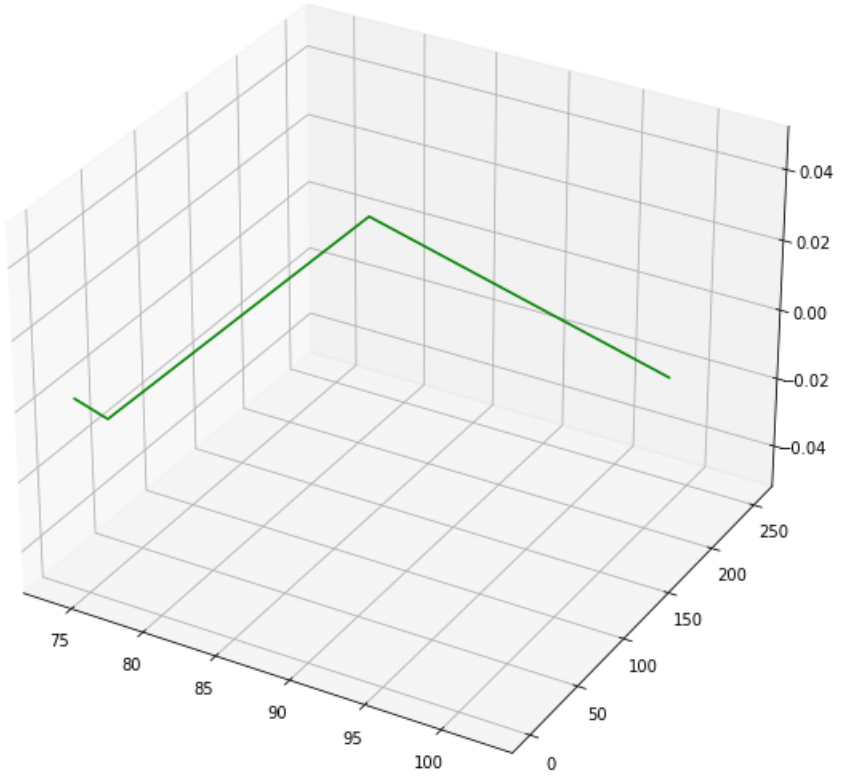
This can be demonstrated by,

In [17]:
```
def get_combined_transformation(x, y, theta, v1, v2):
    x1 = x * math.cos(theta) - y * math.sin(theta) + v1 * time.time()
    time.sleep(0.1)
    y1 = x * math.sin(theta) + y * math.cos(theta) + v2 * time.time()
    return x1 % 255, y1 % 255
```

In [18]:
```
points = [get_combined_transformation(each, each, 1.56, 4, 7) for each in range(50)]
xt, yt = zip(*points)
```

In [20]:
```
plt.figure(figsize = (10,10))
plt.axes(projection='3d')
plt.plot(xt, yt, color="green")
```

Out[20]: [<mpl_toolkits.mplot3d.art3d.Line3D at 0x7f410dd8b430>]



LOGISTIC MAP

The Logistic map is a discrete-time analog of the logistic equation for population growing. Mathematically, the Logistic map is defined as,

$$x_{i+1} = 4px_i(1 - x_i),$$

where parameter p is within the range of [0, 1]. When $p \in [0.89, 1]$, Logistic map is chaotic.

Bifurcation diagram shows the output distribution of a chaotic map along its control parameter, while iteration function describes the output distributions along its inputs. It is noticed that the outputs of Logistic map distribute in a larger area when p approaches to 1.

This can be demonstrated by,

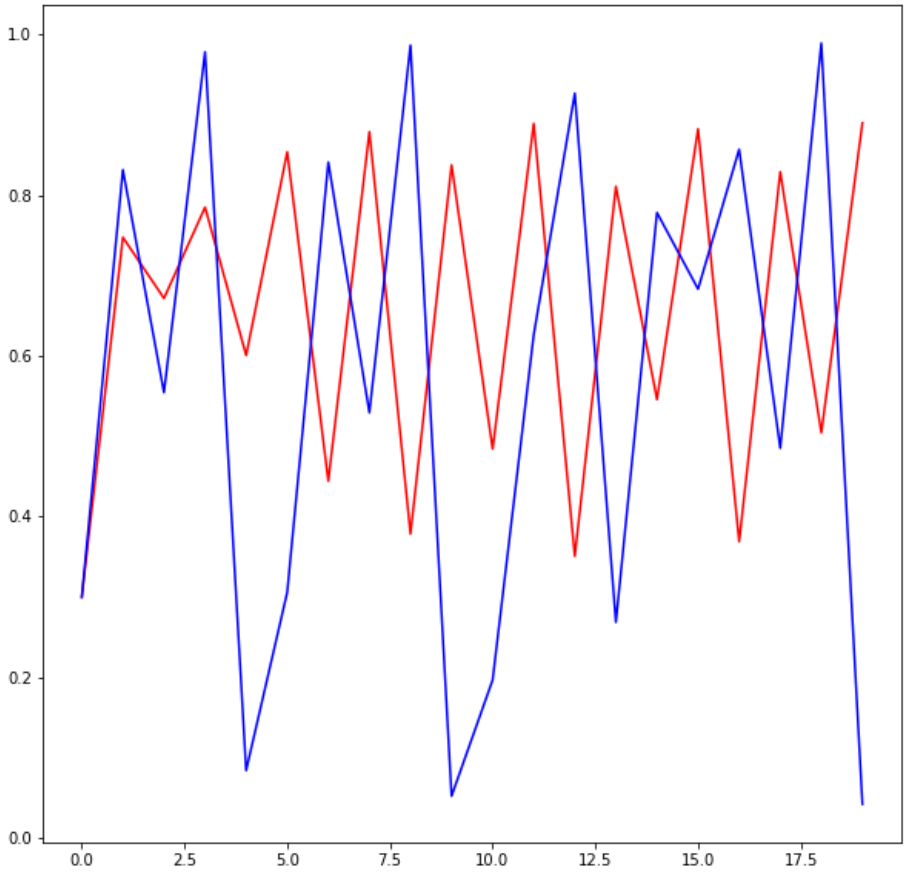
In [11]:
```
def get_logistic_map(current, p, limit):
    def next_element(x, p):
        return 4 * p * x * (1 - x)
    result = []
    for i in range(limit):
        result.append(current)
        current = next_element(current, p)
    return result
```

In [12]:
```
def get_logistic_map_plot_arrays(current, p, limit):
    return np.array([each for each in range(limit)]), np.array(get_logistic_map(current, p, limit))
```

In [13]:
```
x1, y1 = get_logistic_map_plot_arrays(0.3, 0.89, 20)
x2, y2 = get_logistic_map_plot_arrays(0.3, 0.99, 20)
```

In [14]:
```
plt.figure(figsize = (10,10))
plt.plot(x1, y1, color="red")
plt.plot(x2, y2, color="blue")
```

Out[14]: [<matplotlib.lines.Line2D at 0x7fce492ffee0>]



SINE MAP

When sine function has inputs within the range of [0, π], its outputs fall into the range of [0, 1]. Sine map is derived from sine function by transforming its inputs into [0, 1]. It is defined as

$$x_{i+1} = s \sin(\pi x_i)$$

where parameter s ∈ [0, 1]. Sine map is chaotic when s ∈ [0.87, 1]. Although Logistic and Sine maps has totally different mathematical definitions, their chaotic behaviors are quite similar.

This can be demonstrated by,

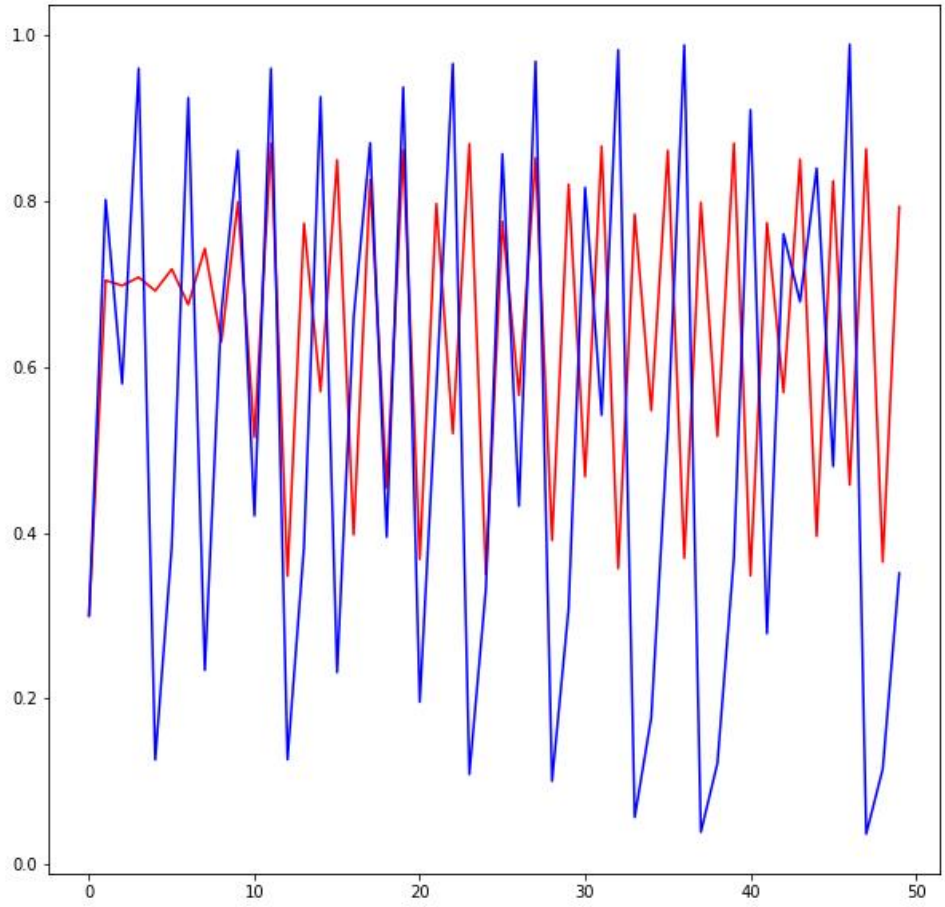
In [15]:
```
def get_sine_map(current, s, limit):
    def next_element(x, s):
        return s * math.sin(math.pi * x)
    result = []
    for i in range(limit):
        result.append(current)
        current = next_element(current, s)
    return result
```

In [16]:
```
def get_sine_map_plot_arrays(current, s, limit):
    return np.array([each for each in range(limit)]), np.array(get_sine_map(current, s, limit))
```

In [17]:
```
x3, y3 = get_sine_map_plot_arrays(0.3, 0.87, 50)
x4, y4 = get_sine_map_plot_arrays(0.3, 0.99, 50)
```

In [18]:
```
plt.figure(figsize = (10,10))
plt.plot(x3, y3, color="red")
plt.plot(x4, y4, color="blue")
```

Out[18]: [<matplotlib.lines.Line2D at 0x7fce4927b400>]



2D-LOGISTIC-ADJUSTED-SINE MAP

We give the mathematical definition of 2D-LASM,

$$x_{i+1} = \sin(\pi\mu(y_i + 3)x_i(1 - x_i))$$
$$y_{i+1} = \sin(\pi\mu(x_{i+1} + 3)y_i(1 - y_i))$$

where parameter $\mu \in [0, 1]$.

2D-LASM is derived from Sine and Logistic maps. The logistic equation $x_i(1 - x_i)$ is first scaled by a factor of μ and fed into the input of Sine map. The phase plane is then extended from 1D to 2D. In 2D-LASM, two inputs are interactively influenced and the output pairs (x_{i+1}, y_{i+1}) distribute into the 2D phase plane.

Compared with Sine and Logistic maps, it has a more complicated structure, and its outputs are more difficult to predict.

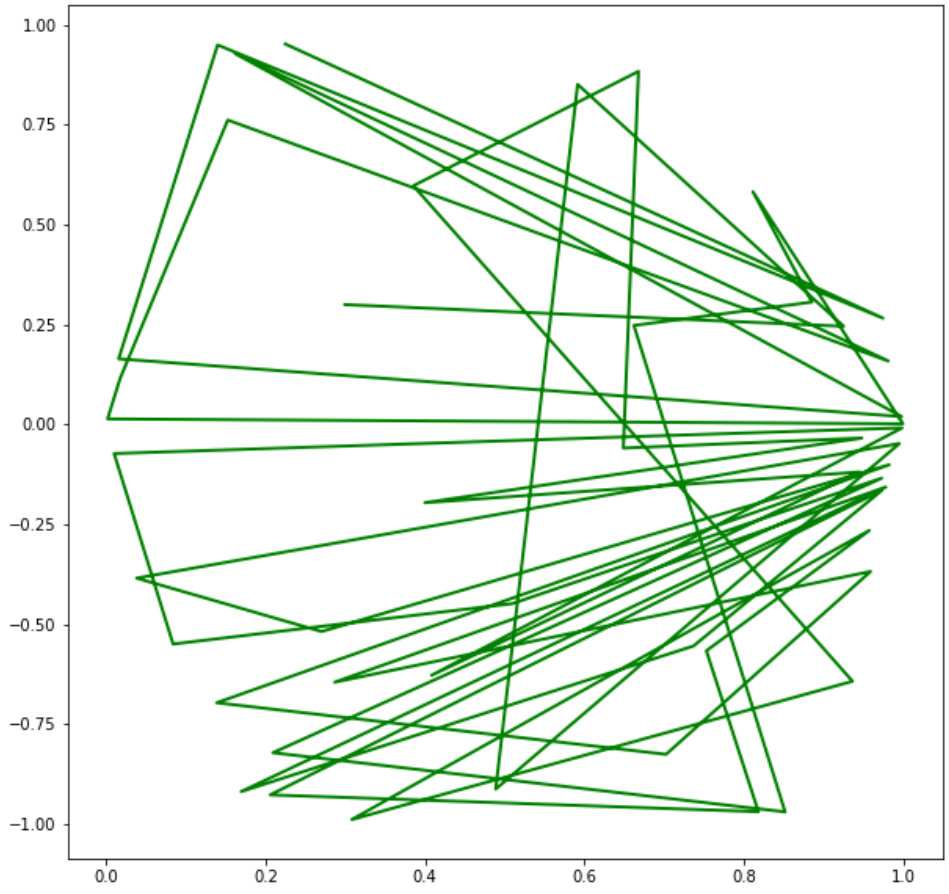
This can be demonstrated by,

In [21]:
```
def get_2D_logistic_adjusted_sine_map(x, y, u, limit):
    def get_next_x(x, y, u):
        return math.sin(math.pi * u * (y + 3) * x * (1 - x))
    def get_next_y(x, y, u):
        return math.sin(math.pi * u * (x + 3) * y * (1 - x))
    result_x = []
    result_y = []
    for i in range(limit):
        result_x.append(x)
        result_y.append(y)
        x = get_next_x(x, y, u)
        y = get_next_y(x, y, u)
    return result_x, result_y
```

In [20]:
```
x5, y5 = get_2D_logistic_adjusted_sine_map(0.3, 0.3, 0.9, 50)
```

In [21]:
```
plt.figure(figsize = (10,10))
plt.plot(x5, y5, color="green", linewidth=2.0)
```

Out[21]: [<matplotlib.lines.Line2D at 0x7fce49260610>]



LINEAR CONGRUENTIAL GENERATOR

The linear congruential generator (LCG) was first published in 1960 by Thomson and Rotenberg. It is one of the best PRNGs that provides fast software and hardware implementations. This is because an LCG requires minimal memory to retain state.

Evidence of the LCG's statistical superiority as a PRNG is that a 96-bit LCG is compliant with (i.e., passes) the most rigorous Big Crush suite. Here, it is used to generate a sequence of integers from 1 to (m − 1), and is mathematically defined as,

$$X_n = (aX_{n-1} - 1 + c) \bmod m$$

where a is a control parameter.

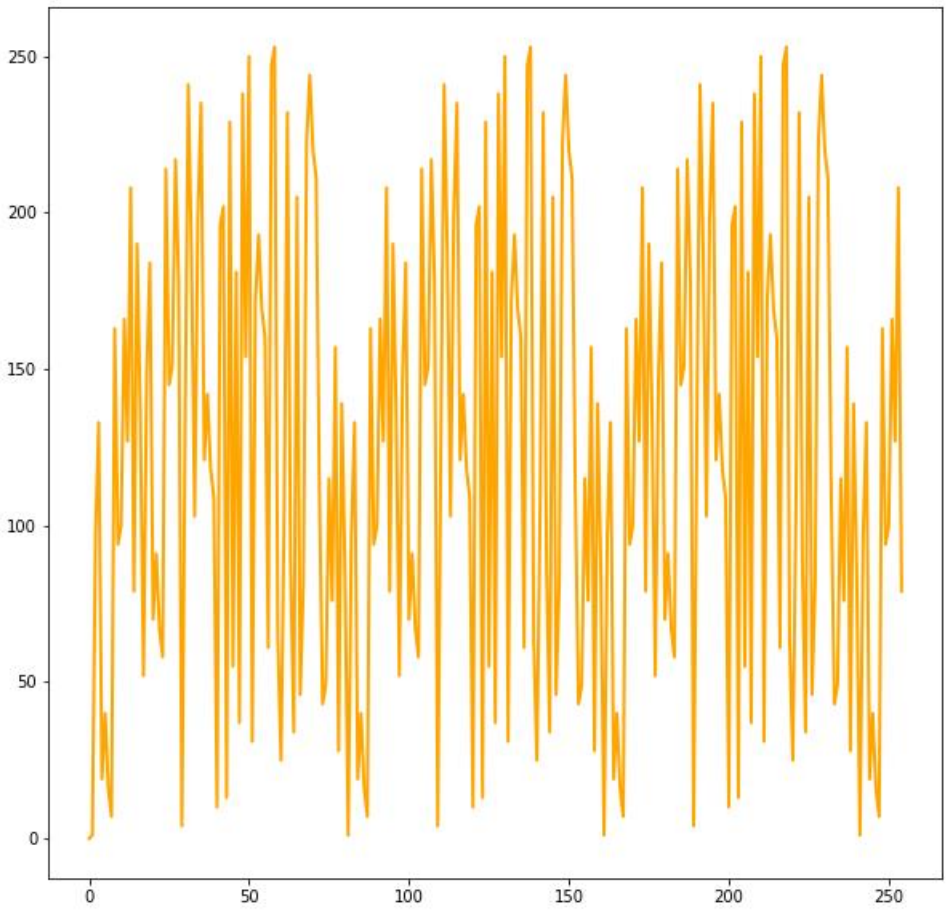
This can be demonstrated by,

In [22]:
```
def linear_congruential_generator(x, a, c, limit):
    def get_next_element(x, a, c, limit):
        return (a * x - 1 + c) % limit
    result = []
    for i in range(limit):
        result.append(x)
        x = get_next_element(x, a, c, limit)
    return result
```

In [23]:
```
x6 = [each for each in range(255)]
y6 = linear_congruential_generator(0, 96, 2, 255)
```

In [24]:
```
plt.figure(figsize = (10,10))
plt.plot(x6, y6, color="orange", linewidth=2.0)
```

Out[24]: [<matplotlib.lines.Line2D at 0x7fce48ffbf10>]



BERNOULLI MAP

The Bernoulli chaotic map is a one-dimensional chaotic map that is defined from the range −A to A. It is one example of strongly chaotic functions that display exponential decay of correlation to their equilibrium values.

Not only is it easy to compute, but also provides much desired properties of chaotic behavior needed for image encryption.

The Lyapunov exponent of the Bernoulli map is $\log_2$. Being an exact system (i.e. mixing and ergodic), after only a small number of iterations, the Bernoulli map when applied with 2 different sets of initial conditions, results in very different trajectories.

It is mathematically defined as,

$$x_{n+1} = \begin{cases} (B\ x_{(n)}) - A, & -A <= x <= 0, \\ (Bx_{(n)}) + A, & 0 <= x <= A, \end{cases}$$

where the sequence is generated with the following parameters: A = 0.5, B = 1.75 and x(1) = (B×0.25)−A. Therefore, the range of the generated chaotic system is from $-1/2$ to $1/2$.

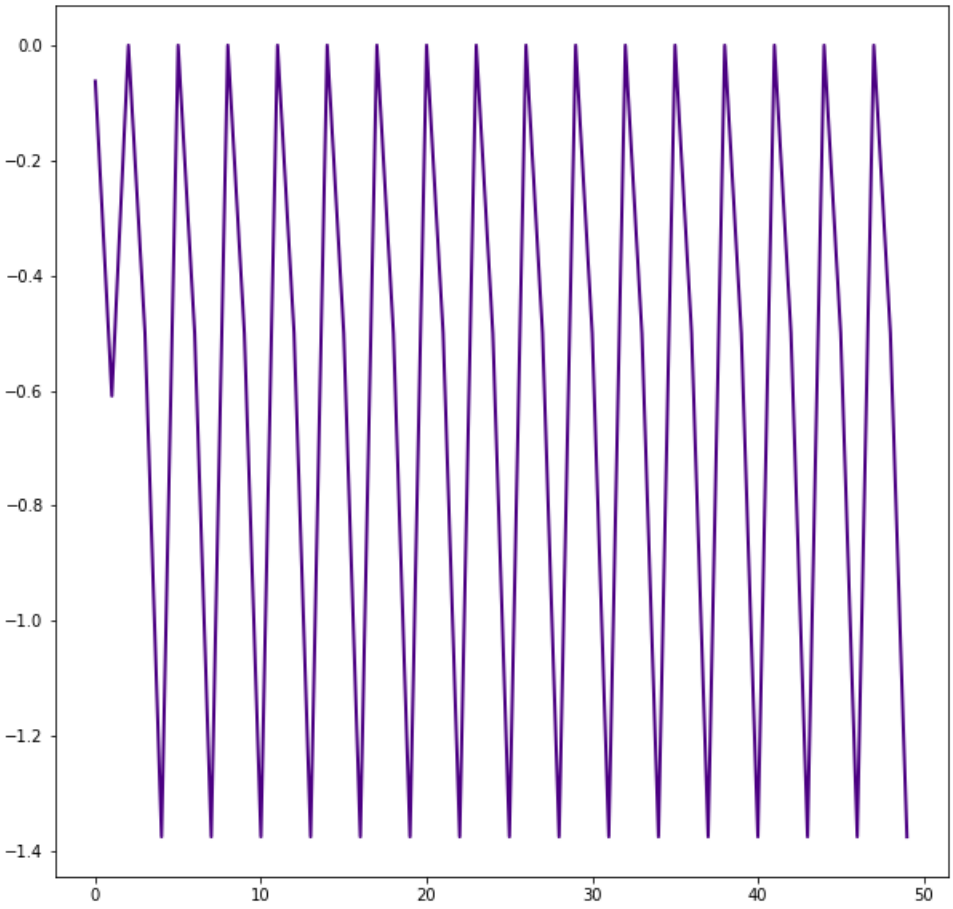
This can be demonstrated by,

In [25]:
```
def get_bernoulli_map(x, a, b, limit):
    def get_next_element(x, a, b):
        if -a <= x <= 0:
            return (b * x) - a
        elif 0 <= x <= a:
            return (b * x) + a
        return 0
    result = []
    for i in range(limit):
        result.append(x)
        x = get_next_element(x, a, b)
    return result
```

In [26]:
```
x7 = [each for each in range(50)]
y7 = get_bernoulli_map((0.25 * 1.75) - 0.5, 0.5, 1.75, 50)
```

In [32]:
```
plt.figure(figsize = (10,10))
plt.plot(x7, y7, color="indigo", linewidth=2.0)
```

Out[32]:
```
[<matplotlib.lines.Line2D at 0x7fce485da340>]
```



TENT MAP

The Tent map is also a chaotic one-dimensional map that displays a good chaotic sequential behavior.

Similar to the Bernoulli map, it has a Lyapunov exponent of $\log_2$, but unlike the Bernoulli map, the x−correlation function for the Tent map is δ−correlated.

It is mathematically expressed as,

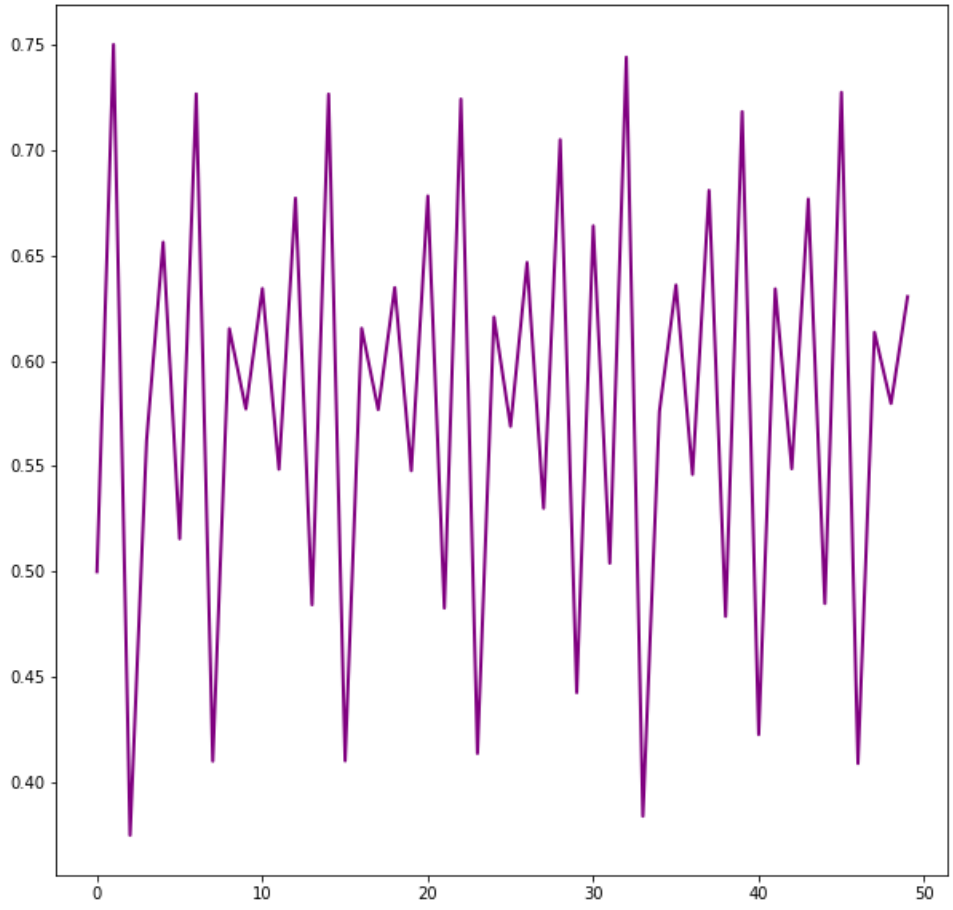
$$x_{n+1} = \begin{cases} C(x_{(n)}), & 0 < x < 1/2, \\ C(1 - x_{(n)}), & 1/2 < x < 1, \end{cases}$$

where the sequence is generated with the parameters
C = 1.5 and x(1) = 0.5.

This can be demonstrated by,

```
In [28]: def get_tent_map(x, c, limit):
             def get_next_element(x, c):
                 if 0 <= x <= 0.5:
                     return c * x
                 if 0.5 <= x <= 1:
                     return c * (1 - x)
                 return x
             result = []
             for i in range(limit):
                 result.append(x)
                 x = get_next_element(x, c)
             return result

In [29]: x8 = [each for each in range(50)]
         y8 = get_tent_map(0.5, 1.5, 50)

In [30]: plt.figure(figsize = (10,10))
         plt.plot(x8, y8, color="purple", linewidth=2.0)
```

Out[30]: [<matplotlib.lines.Line2D at 0x7fce48f4dd00>]



ENCRYPTION ALGORITHM

The proposed image encryption algorithm is implemented over a number of steps, as follows.

- A color image of dimensions N ×N, where for example, N = 256, is loaded and color-separated into its 3 RGB channels.
- Each channel's pixels are then shuffled using the sequence generated from the KAA map, thus inducing diffusion in the image pixels.
- Two encryption keys are generated. The first key is generated using the 2D Sine Logistic Map and the Linear Congruential Generator. It consists of a 256 × 256 bits matrix.
  - The floating-point output sequences, x and y, resulting from the 2D Sine Logistic Map are then converted to 64–bit by utilizing the IEEE–754 double precision conversion. The control parameter a is converted to 64–bit in the same manner. Nevertheless, it is shortened, such that only the first 44 bits are employed.
  - Two sequences, L_1 and L_2, are generated through the use of the linear congruential generator. These employ the parameters, respectively:

    $L1_0 = 3{,}538{,}644{,}446$, $a_L1 = 27$, $c = 0$, $m = 2^{52} - 1$ and
    $L2_0 = 2{,}700{,}000$, $a_L2 = 37$, $c = 1$, $m = 2^{32} - 1$.

    The generated integer sequences L_1 and L_2 are converted to 52 and 32 bits, respectively.

- Hence, the first key is the concatenation of the bits of the x, y, a, L_1 and L_2 which is 64 + 64 + 44 + 52 + 32 resulting in a 256 × 256 bits matrix.

❖ Each channel's shuffled pixels bits are reshaped to 8 × [256 × 256] resulting in 8 sub matrices in each channel to be XORed with the first key generated in a bit–by–bit manner.

❖ The Bernoulli and the Tent chaotic maps are then employed to generate the second key. This key is made up of a 65536 × 8 matrix.

- The decimal number sequence generated from the Bernoulli map is binarized through a simple decision rule. This is carried out by setting a threshold and comparing each element against it. If an element is greater than the threshold, it is assigned a value of 1, otherwise it is assigned a value of 0. The threshold is set to be 0. Furthermore, the length of the output bit sequence is 32768 × 8.
- In a similar fashion a bit sequence is obtained from the Tent map. However, the threshold is set to have a value of 0.5. The length of the output bit sequence is 32768 × 8.
- The second key is the concatenation of both sequences. This generates a 65536 × 8-bit matrix which is XORed again in a bit–by–bit manner with each of the RGB channels of the image.

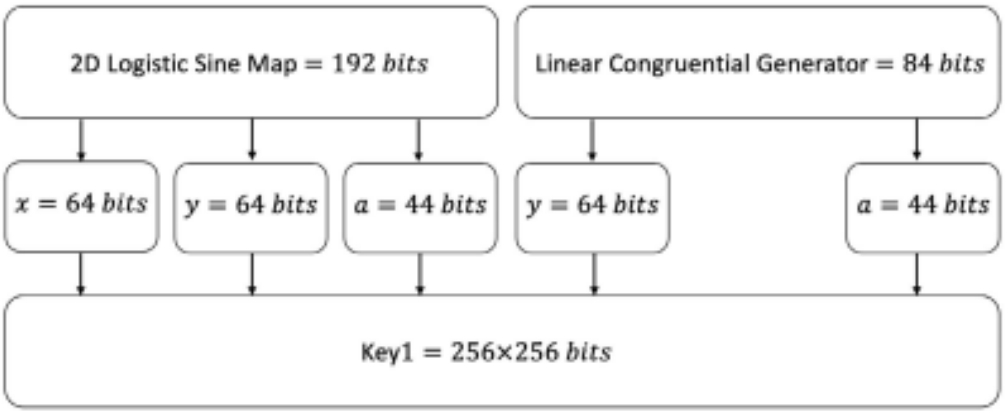


FIGURE 1. **Flow chart showing the generation of the first encryption key, based on the 2D LSM and the LCM.**

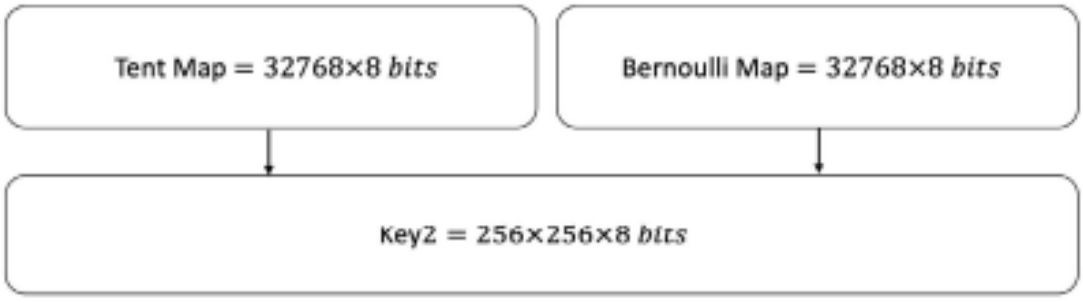


FIGURE 2. **Flow chart showing the generation of the second encryption key, based on the Tent and Bernoulli chaotic maps.**

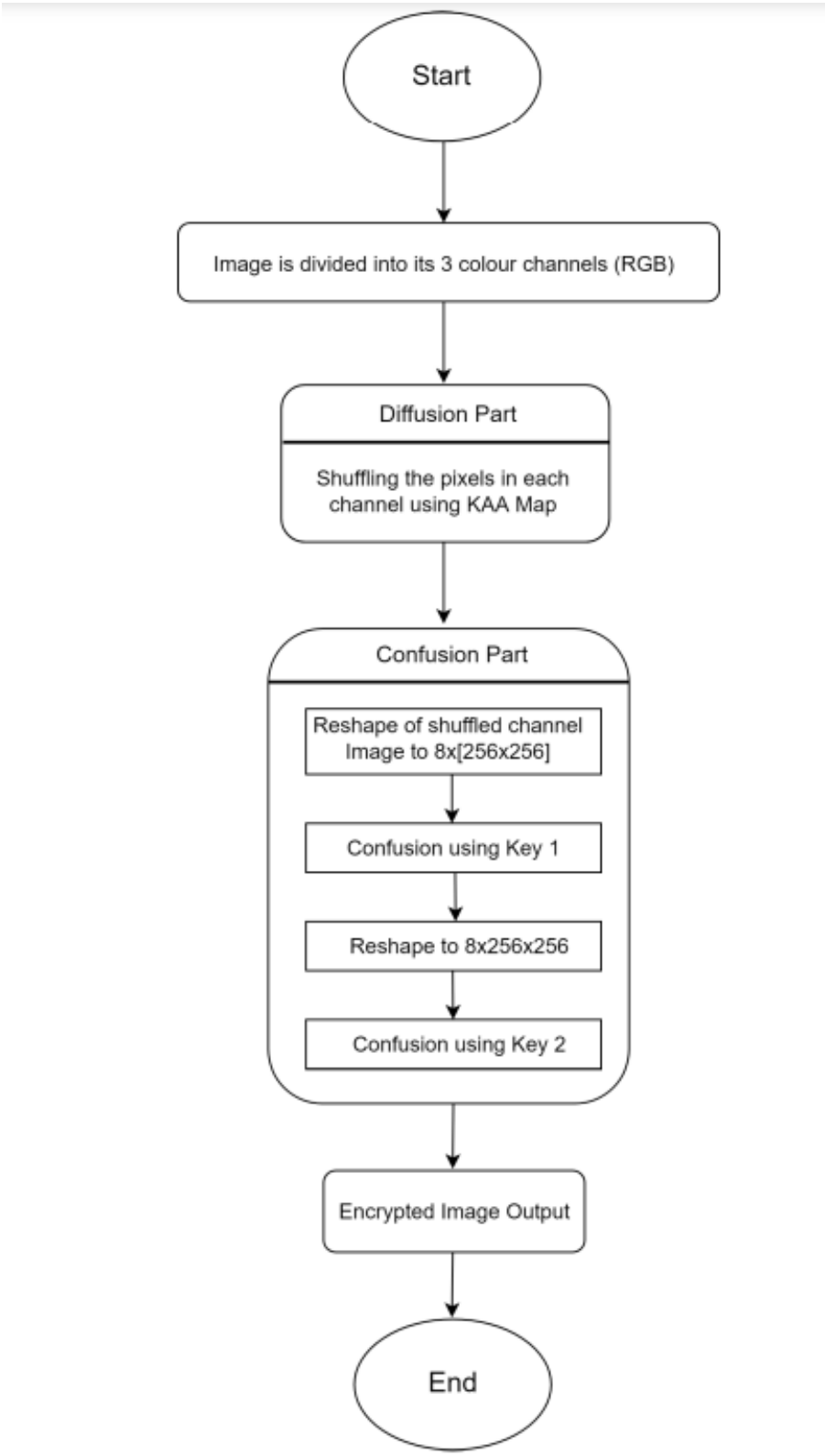
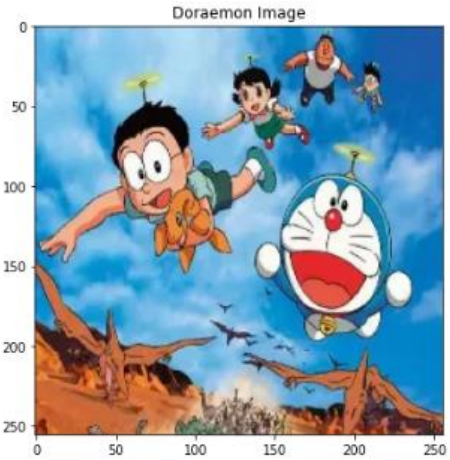
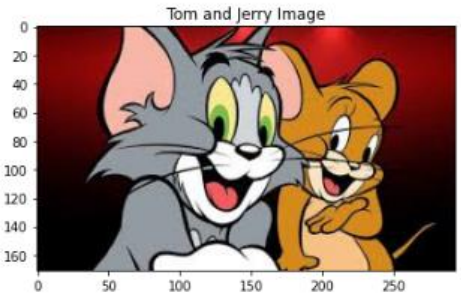


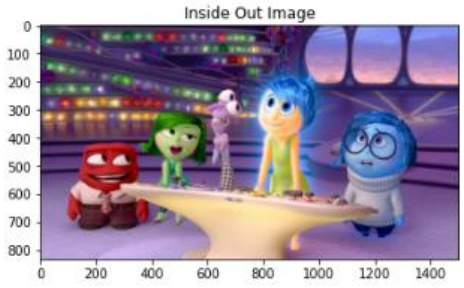
FIGURE 3. **Flow chart showing the encryption process.**

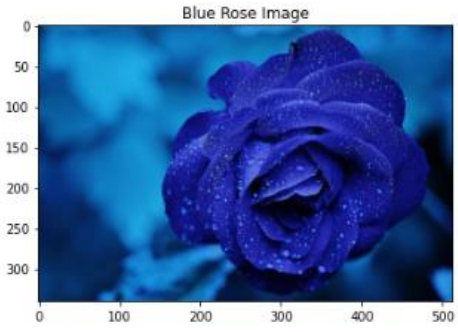
SECURITY ANALYSIS AND NUMERICAL RESULTS

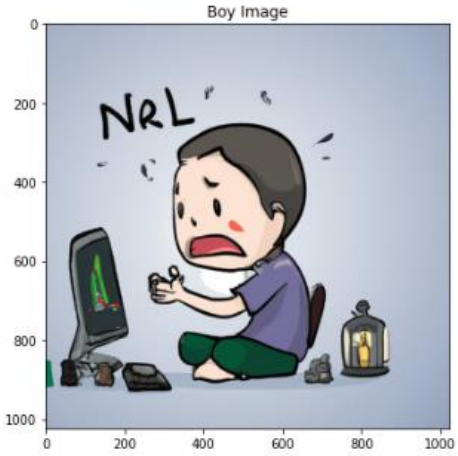
Here in this section, we have tried to analyze our encryption algorithm using standard analysis techniques. For the same purpose we have taken 9 images of different dimensions and nature to compare the efficiency of the entire process.

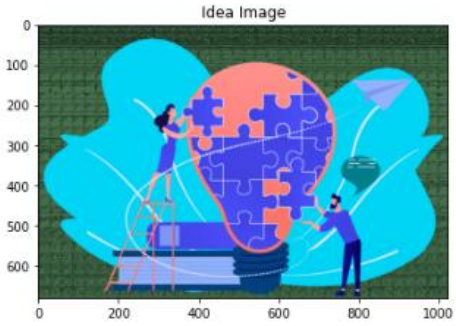


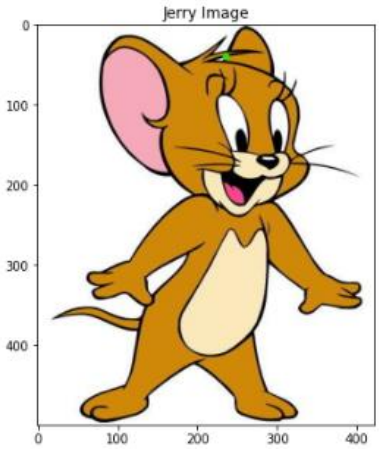


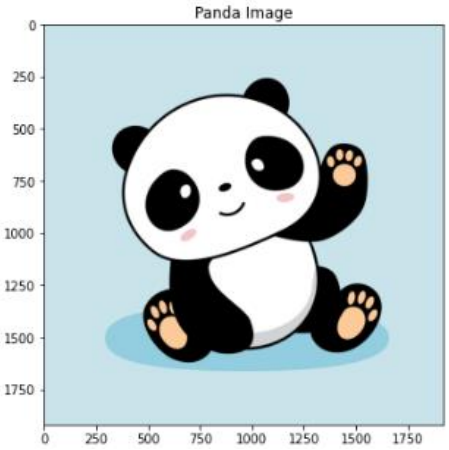







Idea Image
0
100
200
300
400
500
600
0
200
400
600
800
1000


Jerry Image
0
100
200
300
400
0
100
200
300
400

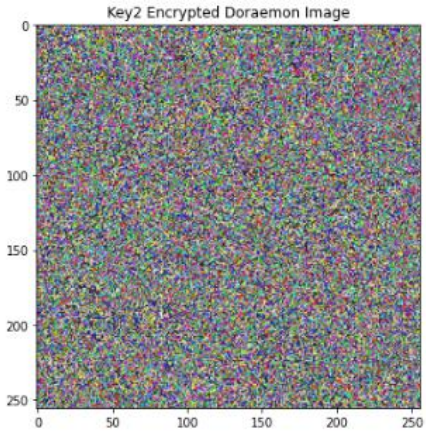

Panda Image
0
250
500
750
1000
1250
1500
1750
0
250
500
750
1000
1250
1500
1750

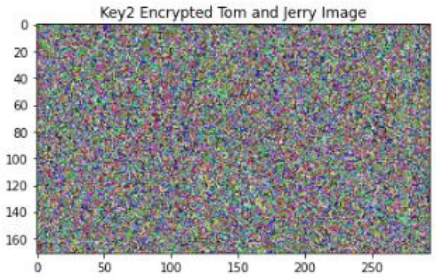
VISUAL AND HISTOGRAM ANALYSIS

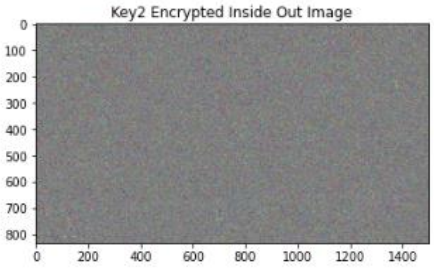
The first and most simple metric for evaluating any image encryption algorithm is through an assessment by the human visual system (HVS).

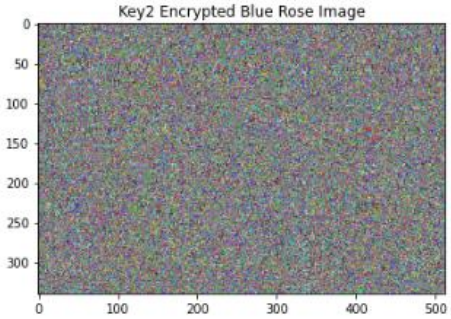
For the above images, we have provided the encrypted images below.

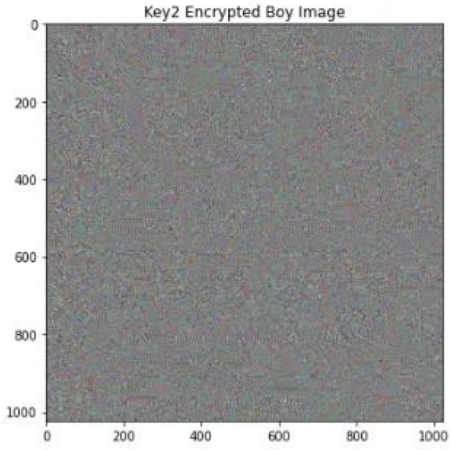
As clear from the visualization it represents unintelligible data.

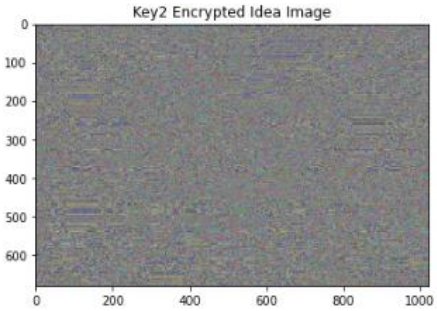


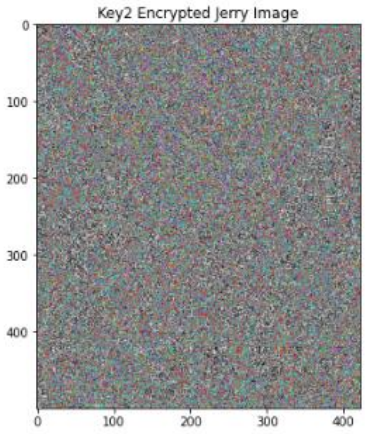


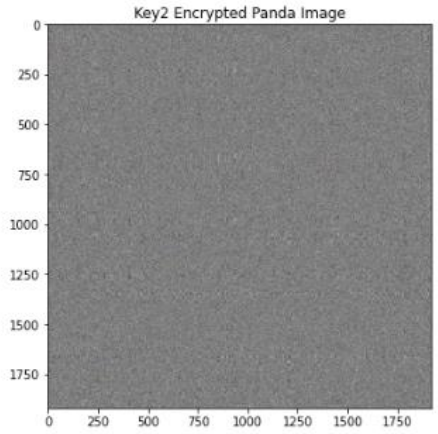










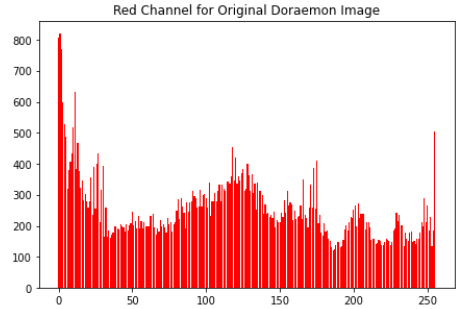


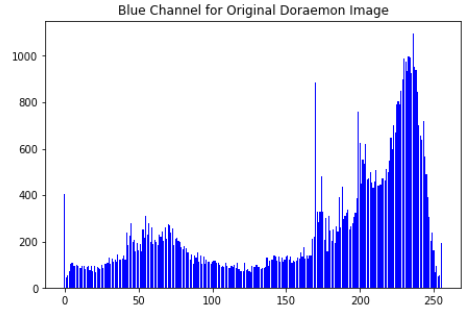
The second visual evaluation metric is the image histogram.

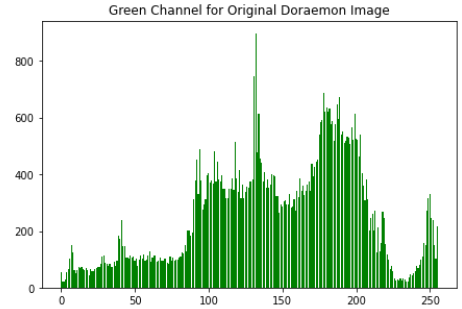
In the context of data encryption, an image histogram is a description of the probability density function for its pixels.

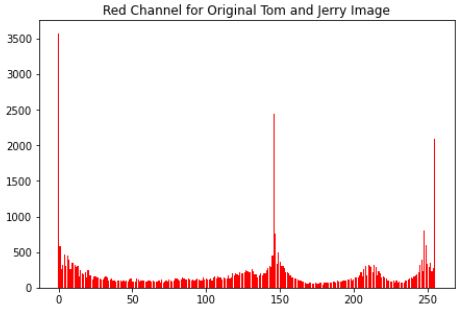
A strong encryption scheme appears in the histogram as a uniform distribution among an image's gray levels, hiding any statistical characteristics in the image.

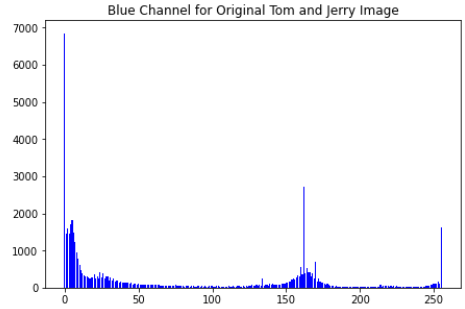
This can be interpreted in the histograms provided below, which proves the difficulty of information retrieval from encrypted images and thus the resistance of the proposed encryption algorithm against statistical attacks.

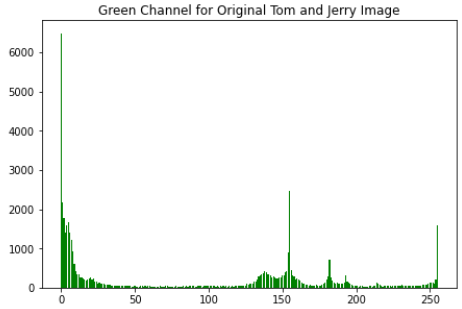

Red Channel for Original Doraemon Image


Blue Channel for Original Doraemon Image

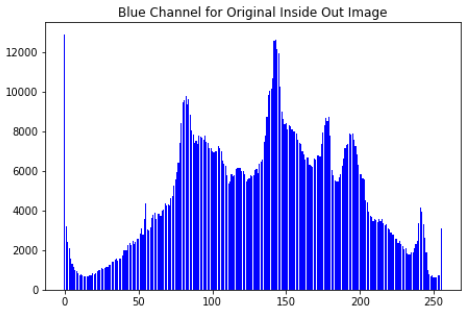

Green Channel for Original Doraemon Image

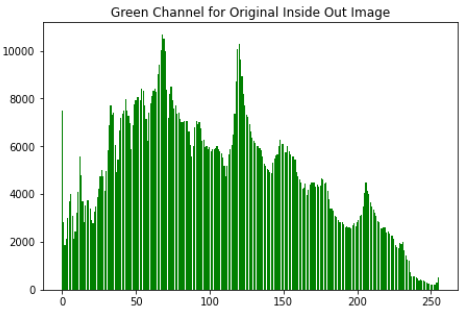

Red Channel for Original Tom and Jerry Image

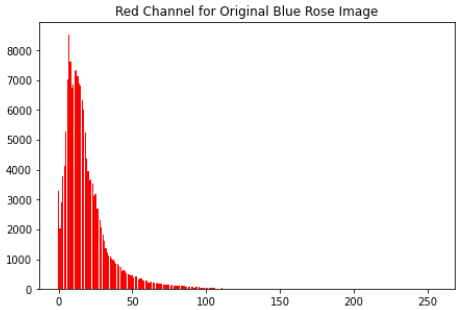

Blue Channel for Original Tom and Jerry Image

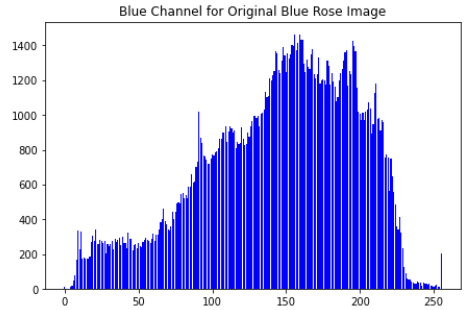

Green Channel for Original Tom and Jerry Image

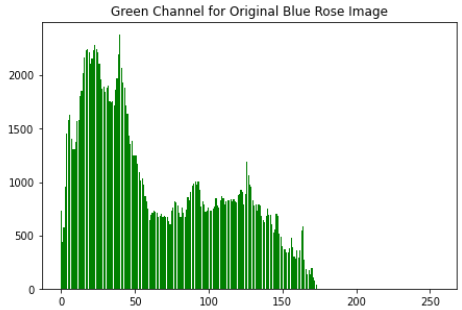

Red Channel for Original Inside Out Image

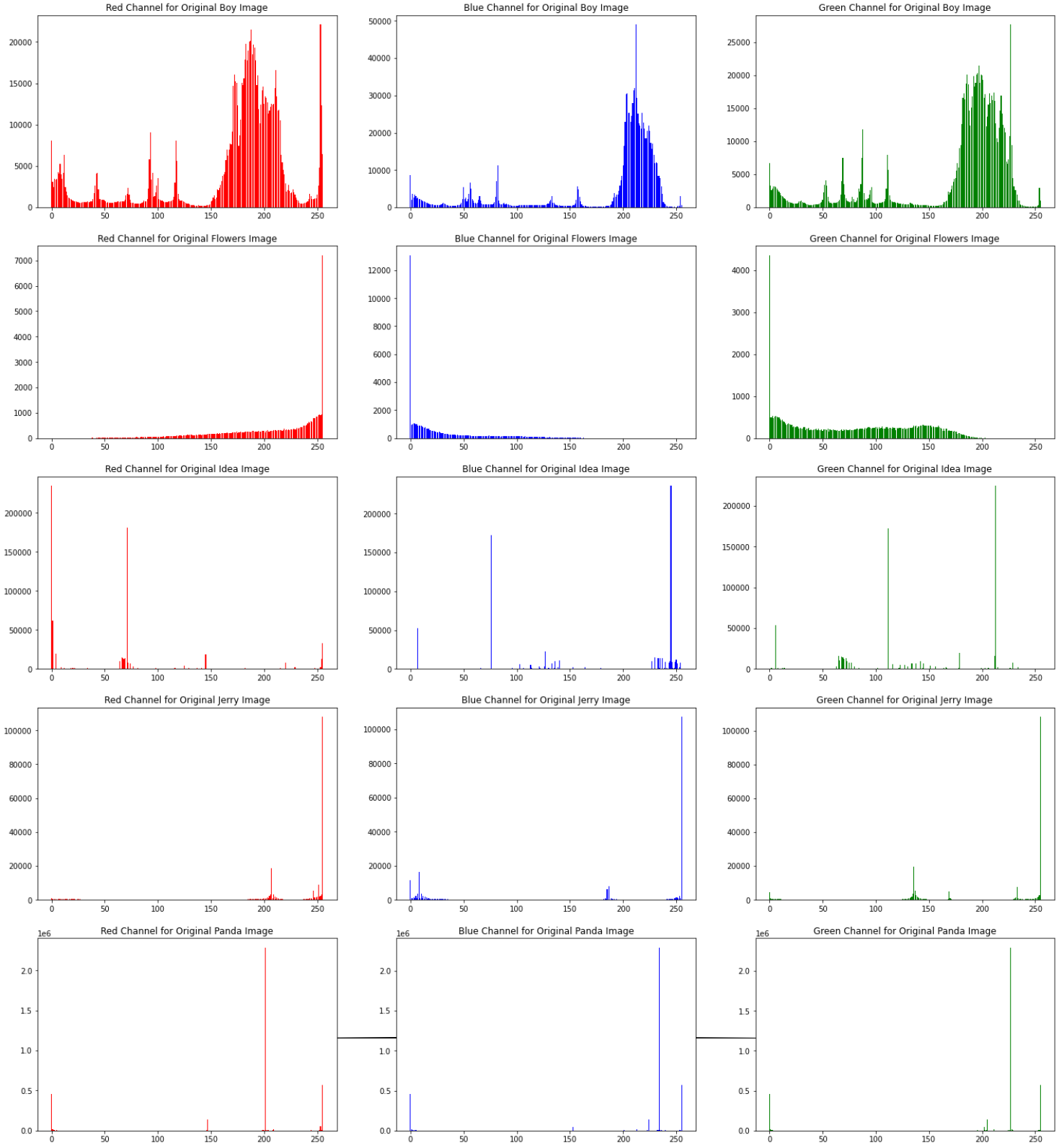

Blue Channel for Original Inside Out Image

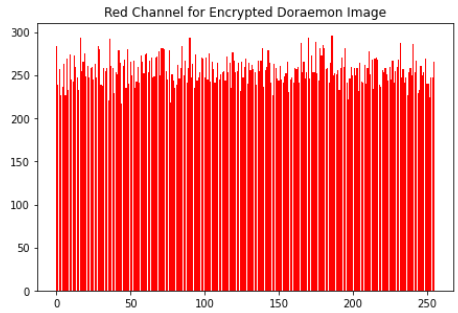

Green Channel for Original Inside Out Image

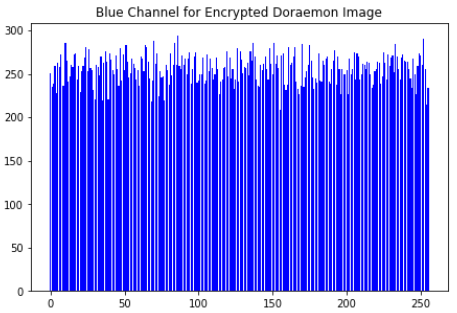

Red Channel for Original Blue Rose Image

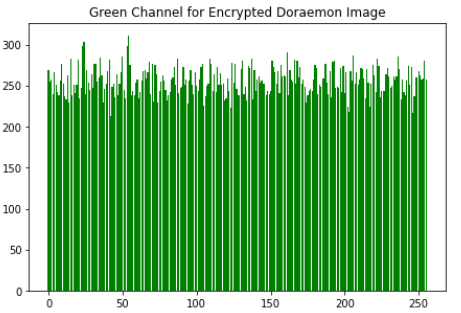

Blue Channel for Original Blue Rose Image

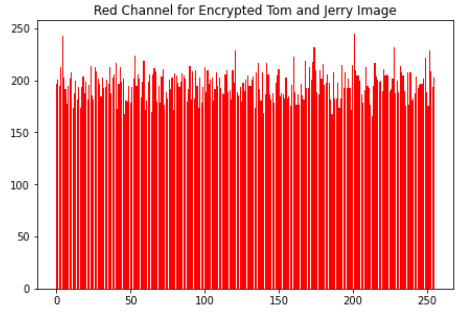

Green Channel for Original Blue Rose Image

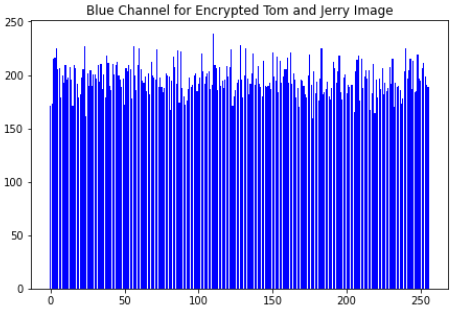

Red Channel for Original Boy Image
Blue Channel for Original Boy Image
Green Channel for Original Boy Image
Red Channel for Original Flowers Image
Blue Channel for Original Flowers Image
Green Channel for Original Flowers Image
Red Channel for Original Idea Image
Blue Channel for Original Idea Image
Green Channel for Original Idea Image
Red Channel for Original Jerry Image
Blue Channel for Original Jerry Image
Green Channel for Original Jerry Image
Red Channel for Original Panda Image
Blue Channel for Original Panda Image
Green Channel for Original Panda Image
1e6

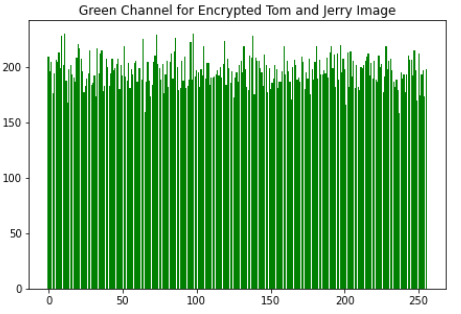

Red Channel for Encrypted Doraemon Image

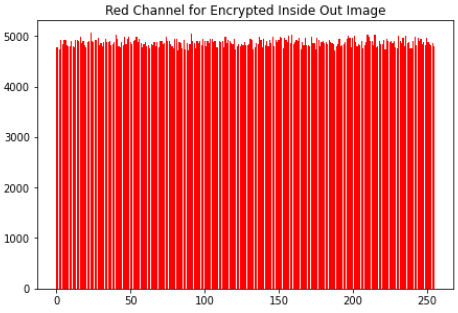

Blue Channel for Encrypted Doraemon Image

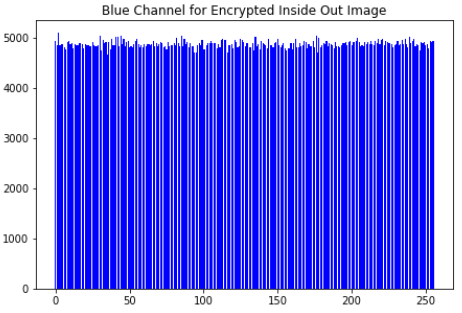

Green Channel for Encrypted Doraemon Image

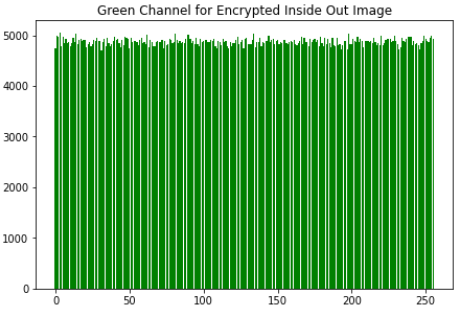

Red Channel for Encrypted Tom and Jerry Image

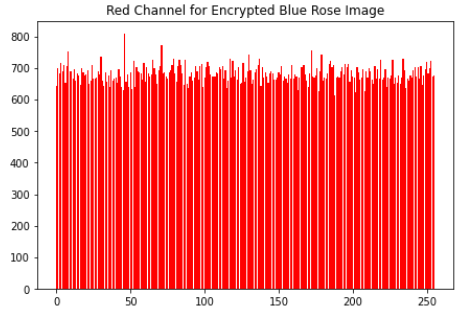

Blue Channel for Encrypted Tom and Jerry Image

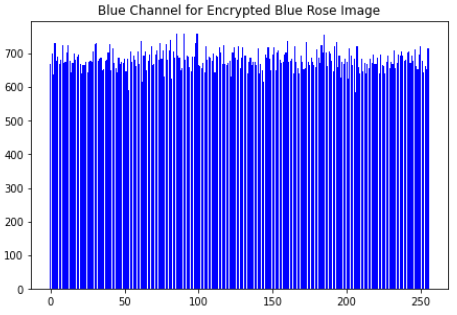

Green Channel for Encrypted Tom and Jerry Image

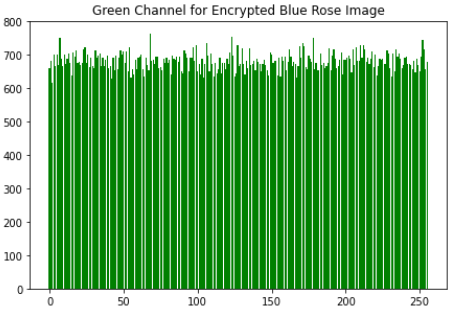

Red Channel for Encrypted Inside Out Image

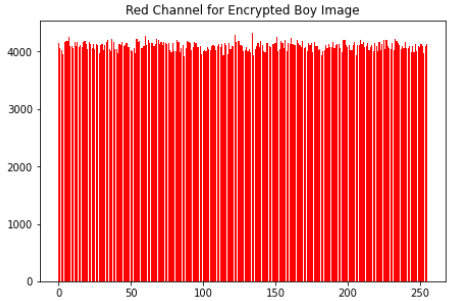

Blue Channel for Encrypted Inside Out Image

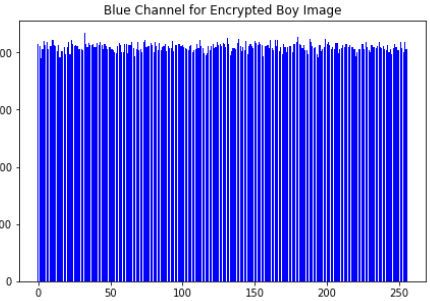

Green Channel for Encrypted Inside Out Image

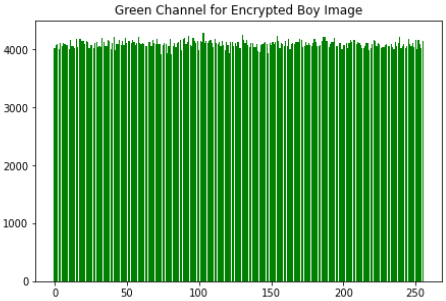

Red Channel for Encrypted Blue Rose Image

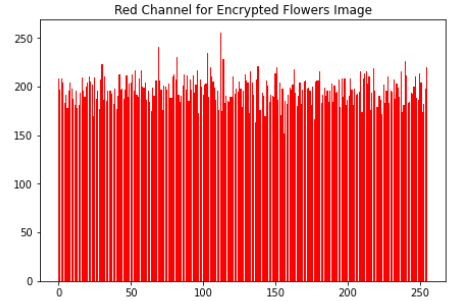

Blue Channel for Encrypted Blue Rose Image

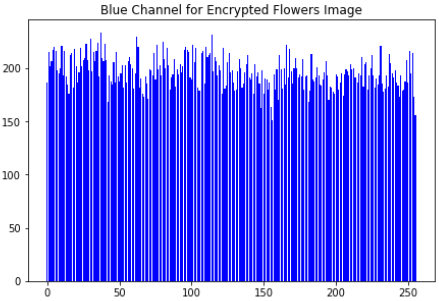

Green Channel for Encrypted Blue Rose Image

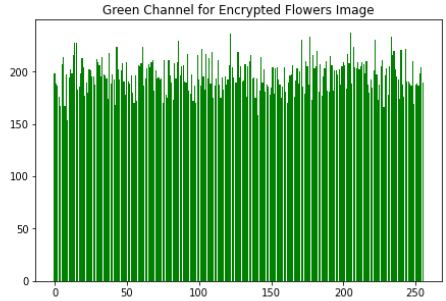

Red Channel for Encrypted Boy Image

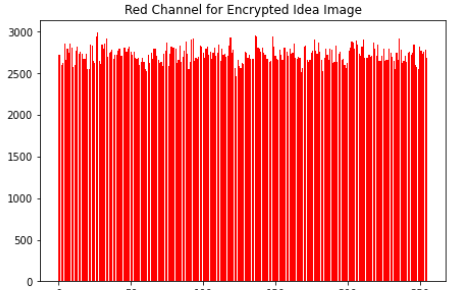

Blue Channel for Encrypted Boy Image

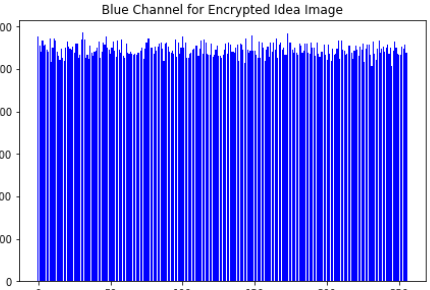

Green Channel for Encrypted Boy Image

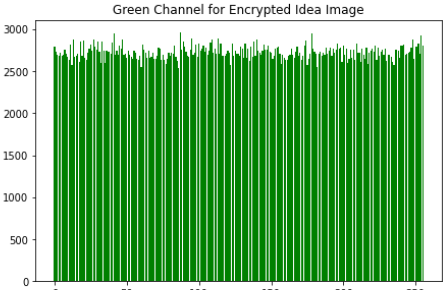

Red Channel for Encrypted Flowers Image

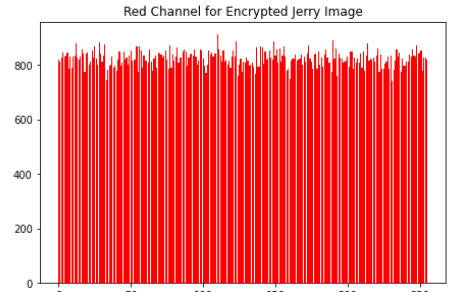

Blue Channel for Encrypted Flowers Image

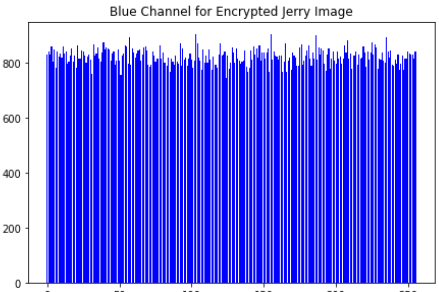

Green Channel for Encrypted Flowers Image

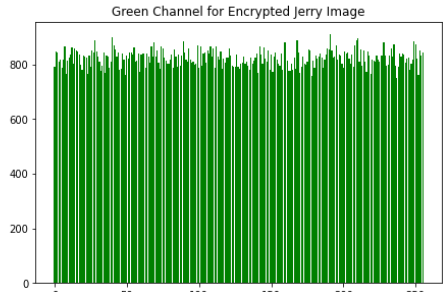

Red Channel for Encrypted Idea Image

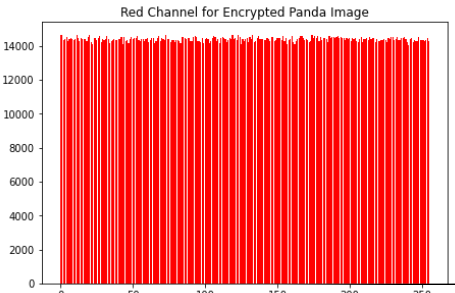

Blue Channel for Encrypted Idea Image

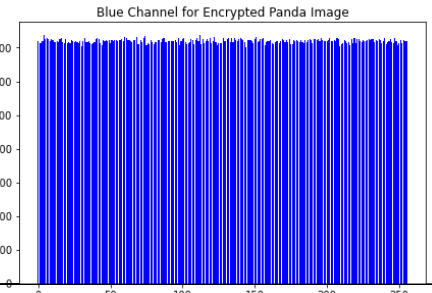

Green Channel for Encrypted Idea Image

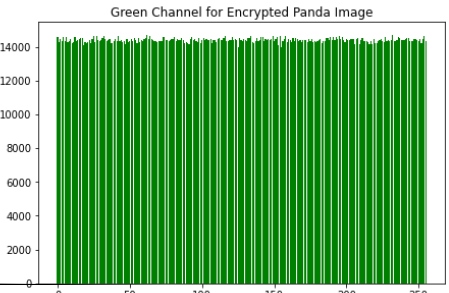

Red Channel for Encrypted Jerry Image


Blue Channel for Encrypted Jerry Image


Green Channel for Encrypted Jerry Image


Red Channel for Encrypted Panda Image


Blue Channel for Encrypted Panda Image


Green Channel for Encrypted Panda Image

INFORMATION ENTROPY

Information entropy is used as a metric to evaluate an image based on the randomness of the distribution of its gray pixels.

The equation shown below demonstrates how it is calculated for an image, where $p(m_i)$ represents the probability of occurrence of each symbol m in the total number of M symbols in an image.

$$H(m) = \sum_{i=1}^{M} p(m_i) log_2 \frac{1}{p(m_i)}$$

Ideally, for a randomly encrypted gray scale image with 256 symbols, its entropy value would be 8, and is reduced with less random encryption.

The below tables show the information entropy of the RGB channels for the 9 images we encrypted. All of the encrypted images show information entropy very close to 8.

```
################################################################################
Encrypted Image
################################################################################
                     Red Channel  Blue Channel  Green Channel     Image
Doraemon Image          7.997365      7.997156       7.996973  7.999091
Tom and Jerry Image     7.996433      7.996289       7.996704  7.998772
Inside Out Image        7.999843      7.999864       7.999865  7.999951
Blue Rose Image         7.998881      7.998763       7.998973  7.999574
Boy Image               7.999782      7.999794       7.999843  7.999937
Flowers Image           7.996473      7.996012       7.995753  7.998907
Idea Image              7.999156      7.999476       7.999436  7.999753
Jerry Image             7.999165      7.999153       7.999032  7.999686
Panda Image             7.999950      7.999950       7.999948  7.999976
################################################################################
Original Image
################################################################################
                     Red Channel  Blue Channel  Green Channel     Image
Doraemon Image          7.899040      7.542754       7.629843  7.927936
Tom and Jerry Image     7.381227      6.404629       6.462237  6.982173
Inside Out Image        7.802691      7.727499       7.793973  7.860580
Blue Rose Image         5.569047      7.621424       7.223039  7.468365
Boy Image               6.970757      6.446514       6.803799  7.054215
Flowers Image           6.709565      5.768534       7.264404  7.336660
Idea Image              3.341149      3.522701       3.652871  4.960531
Jerry Image             3.910089      3.789968       3.813268  4.212389
Panda Image             1.990224      1.988982       1.970239  3.054637
################################################################################
```

MEAN SQUARED ERROR

The Mean Squared Error (MSE) is one of the popular metrics used for evaluating the error between original and encrypted images in order to assess the reliability of any encryption algorithm.

It is calculated as shown in the equation below by squaring the difference between the pixel of the plain image $P_{(i,j)}$ and the encrypted image $E_{(i,j)}$ for all M ×N pixels in the image, then dividing the total squared error for all pixels by their count. Higher MSE values indicate better resilience of an encryption algorithm against statistical attacks.

$$MSE = \frac{\sum_{i=0}^{M-1}\sum_{j=0}^{n-1}(P_{(i,j)} - E_{(i,j)})^2}{M\ X\ N}$$

The table below demonstrates Mean Squared Error for the 9 encrypted images.

```
################################################################################
Mean Squared Error
################################################################################
                            MSE
Doraemon Image       32294.451660
Tom and Jerry Image  42382.251542
Inside Out Image     27786.351954
Blue Rose Image      37549.497007
Boy Image            34731.964033
Flowers Image        42296.647968
Idea Image           40522.690314
Jerry Image          54125.648346
Panda Image          48954.977058
################################################################################
```

PEAK SIGNAL TO NOISE RATIO

Based on the MSE, the Peak Signal to Noise Ratio (PSNR) is another performance metric that evaluates the quality of encryption algorithms.

As shown in the equation below, the PSNR is calculated by evaluating the log value of the ratio between the square of the maximum pixel value I_{max} to the MSE, where I_{max} is ideally equal to 255.

Since PSNR and MSE are inversely proportional, lower PSNR values indicate better resilience of an encryption algorithm.

$$PSNR = 10log\left(\frac{I_{max}^2}{MSE}\right)$$

The below table shows the PSNR values of our encrypted images,

```
################################################################################
Peak Signal To Noise Ratio
################################################################################
                         PSNR
Doraemon Image       7.810737
Tom and Jerry Image  6.630176
Inside Out Image     8.463701
Blue Rose Image      7.155975
Boy Image            7.494723
Flowers Image        6.638957
Idea Image           6.825033
Jerry Image          5.567985
Panda Image          6.004048
################################################################################
```

CORRELATION COEFFICIENT ANALYSIS

Analysis of correlation coefficient, r, is another metric that is highly important in evaluating the performance of an encryption algorithm.

The correlation coefficient between 2 pixels x and y is calculated as shown in the equations below; resulting in a value that varies between −1 and 1.

For an encryption scheme to be resilient against statistical attacks, the correlation coefficient between adjacent pixels should be close to zero to indicate no correlation. Otherwise, values close to −1 indicate strong negative correlation and values close to 1 indicate strong positive correlation, which make it feasible for an encrypted image to be exposed by statistical attacks.

$$r_{xy} = \frac{cov\,(x,y)}{\sqrt{D(x)}\sqrt{D(y)}}$$

$$cov\,(x,y) = \frac{1}{N}\sum_{i=1}^{N}(x_i - E(x))(y_i - E(y))$$

$$D(x) = \frac{1}{N}\sum_{i=1}^{N}(x_i - E(x))^2$$

$$E(x) = \frac{1}{N}\sum_{i=1}^{N}(x_i)$$

The below table shows the correlation coefficient analysis between the original and encrypted images of the 9 sample images we have considered.

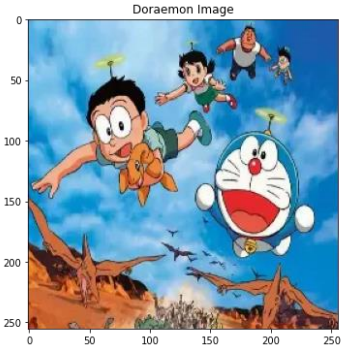
```
##########################################################################################
Correlation Coefficient Analysis
##########################################################################################
                    correlation coefficient
Doraemon Image                     0.001120
Tom and Jerry Image               -0.006055
Inside Out Image                   0.000047
Blue Rose Image                   -0.002505
Boy Image                         -0.001050
Flowers Image                     -0.009307
Idea Image                        -0.000402
Jerry Image                       -0.002517
Panda Image                       -0.000469
##########################################################################################
```

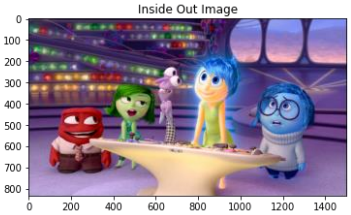
CODE

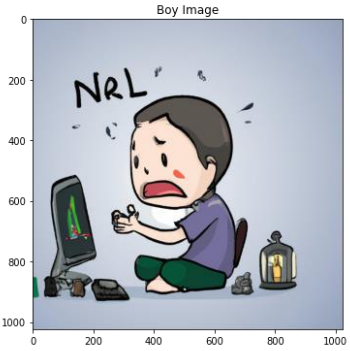
```
import cv2
import math
import time
import random
import pandas            as pd
import numpy             as np
import matplotlib.pyplot as plt
from   scipy.stats       import entropy
from   matplotlib        import rcParams

# Path to images for encrypting
doraemon_image_path      = '../Pictures/doraemon.webp'
tom_and_jerry_image_path = '../Pictures/tom_and_jerry.jpg'
inside_out_image_path    = '../Pictures/inside_out.jpg'
blue_rose_image_path     = '../Pictures/blue_rose.jpg'
boy_image_path           = '../Pictures/boy.jpg'
flowers_image_path       = '../Pictures/flowers.jpg'
idea_image_path          = '../Pictures/idea.png'
jerry_image_path         = '../Pictures/jerry.jpg'
panda_image_path         = '../Pictures/panda.webp'

# Reading the images for further processing
doraemon_image      = cv2.imread(doraemon_image_path)
tom_and_jerry_image = cv2.imread(tom_and_jerry_image_path)
inside_out_image    = cv2.imread(inside_out_image_path)
blue_rose_image     = cv2.imread(blue_rose_image_path)
boy_image           = cv2.imread(boy_image_path)
flowers_image       = cv2.imread(flowers_image_path)
idea_image          = cv2.imread(idea_image_path)
jerry_image         = cv2.imread(jerry_image_path)
panda_image         = cv2.imread(panda_image_path)

# Splitting the different images into red, blue and green images
# cv2 splits the images in order of blue, green and red channels
# while it merges the images in the order of red, green and blue channels
blue_doraemon_image,      green_doraemon_image,      red_doraemon_image
= cv2.split(np.array(doraemon_image))
blue_tom_and_jerry_image, green_tom_and_jerry_image, red_tom_and_jerry_image
= cv2.split(np.array(tom_and_jerry_image))
blue_inside_out_image,    green_inside_out_image,    red_inside_out_image
= cv2.split(np.array(inside_out_image))
blue_blue_rose_image,     green_blue_rose_image,     red_blue_rose_image
= cv2.split(np.array(blue_rose_image))
blue_boy_image,           green_boy_image,           red_boy_image
```

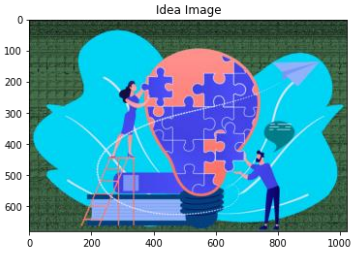
```
= cv2.split(np.array(boy_image))
blue_flowers_image,       green_flowers_image,       red_flowers_image
= cv2.split(np.array(flowers_image))
blue_idea_image,          green_idea_image,          red_idea_image
= cv2.split(np.array(idea_image))
blue_jerry_image,         green_jerry_image,         red_jerry_image
= cv2.split(np.array(jerry_image))
blue_panda_image,         green_panda_image,         red_panda_image
= cv2.split(np.array(panda_image))

# Getting the dimensions of the differnt images
x_doraemon_image,      y_doraemon_image,      z_doraemon_image      = np.shap
e(doraemon_image)
x_tom_and_jerry_image, y_tom_and_jerry_image, z_tom_and_jerry_image = np.shap
e(tom_and_jerry_image)
x_inside_out_image,    y_inside_out_image,    z_inside_out_image    = np.shap
e(inside_out_image)
x_blue_rose_image,     y_blue_rose_image,     z_blue_rose_image     = np.shap
e(blue_rose_image)
x_boy_image,           y_boy_image,           z_boy_image           = np.shap
e(boy_image)
x_flowers_image,       y_flowers_image,       z_flowers_image       = np.shap
e(flowers_image)
x_idea_image,          y_idea_image,          z_idea_image          = np.shap
e(idea_image)
x_jerry_image,         y_jerry_image,         z_jerry_image         = np.shap
e(jerry_image)
x_panda_image,         y_panda_image,         z_panda_image         = np.shap
e(panda_image)

# Merging the three different channels to get the image
rgb_doraemon_image       = cv2.merge([red_doraemon_image,      green_doraemon
_image,      blue_doraemon_image])
rgb_tom_and_jerry_image  = cv2.merge([red_tom_and_jerry_image, green_tom_and_
jerry_image, blue_tom_and_jerry_image])
rgb_inside_out_image     = cv2.merge([red_inside_out_image,    green_inside_o
ut_image,    blue_inside_out_image])
rgb_blue_rose_image      = cv2.merge([red_blue_rose_image,     green_blue_ros
e_image,     blue_blue_rose_image])
rgb_boy_image            = cv2.merge([red_boy_image,           green_boy_imag
e,           blue_boy_image])
rgb_flowers_image        = cv2.merge([red_flowers_image,       green_flowers_
image,       blue_flowers_image])
rgb_idea_image           = cv2.merge([red_idea_image,          green_idea_ima
ge,          blue_idea_image])
rgb_jerry_image          = cv2.merge([red_jerry_image,         green_jerry_im
```

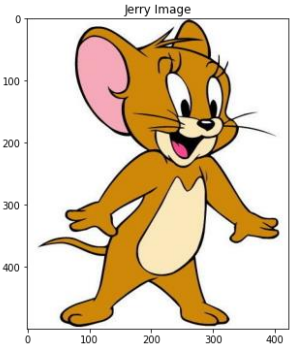
```
age,         blue_jerry_image])
rgb_panda_image          = cv2.merge([red_panda_image,         green_panda_im
age,         blue_panda_image])

# Function to display images horizontally in IPython Cell
# length             - Length of the figure
# breadth            - Breadth of the figure
# rows               - Number of rows to be there in the plot
# columns            - Number of columns to be there in the plot
# images_list        - List of images to be displayed
# images_titles_list - list of titles to the images to be displayed
def display_images_horizontaly(length, breadth, rows, columns, images_list, i
mages_titles_list):
    rcParams['figure.figsize'] = length, breadth
    figure, axis = plt.subplots(rows, columns)
    for i in range(rows):
        for j in range(columns):
            one_dimensional_index = rows * i + j
            axis[i][j].set_title(images_titles_list[one_dimensional_index])
            axis[i][j].imshow(images_list[one_dimensional_index])
    return None

# Creating list of all the images and images titles
images_list        = [rgb_doraemon_image, rgb_tom_and_jerry_image, rgb_inside
_out_image, rgb_blue_rose_image, rgb_boy_image,
                      rgb_flowers_image, rgb_idea_image, rgb_jerry_image, rgb
_panda_image]
images_titles_list = ["Doraemon Image", "Tom and Jerry Image", "Inside Out Im
age", "Blue Rose Image", "Boy Image",
                      "Flowers Image", "Idea Image", "Jerry Image", "Panda Im
age"]

display_images_horizontaly(20, 20, 3, 3, images_list, images_titles_list)
```

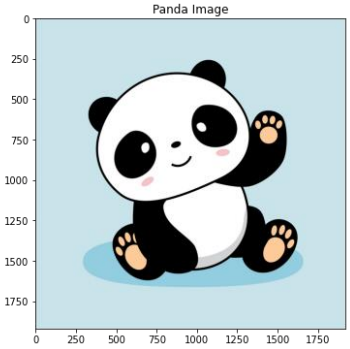










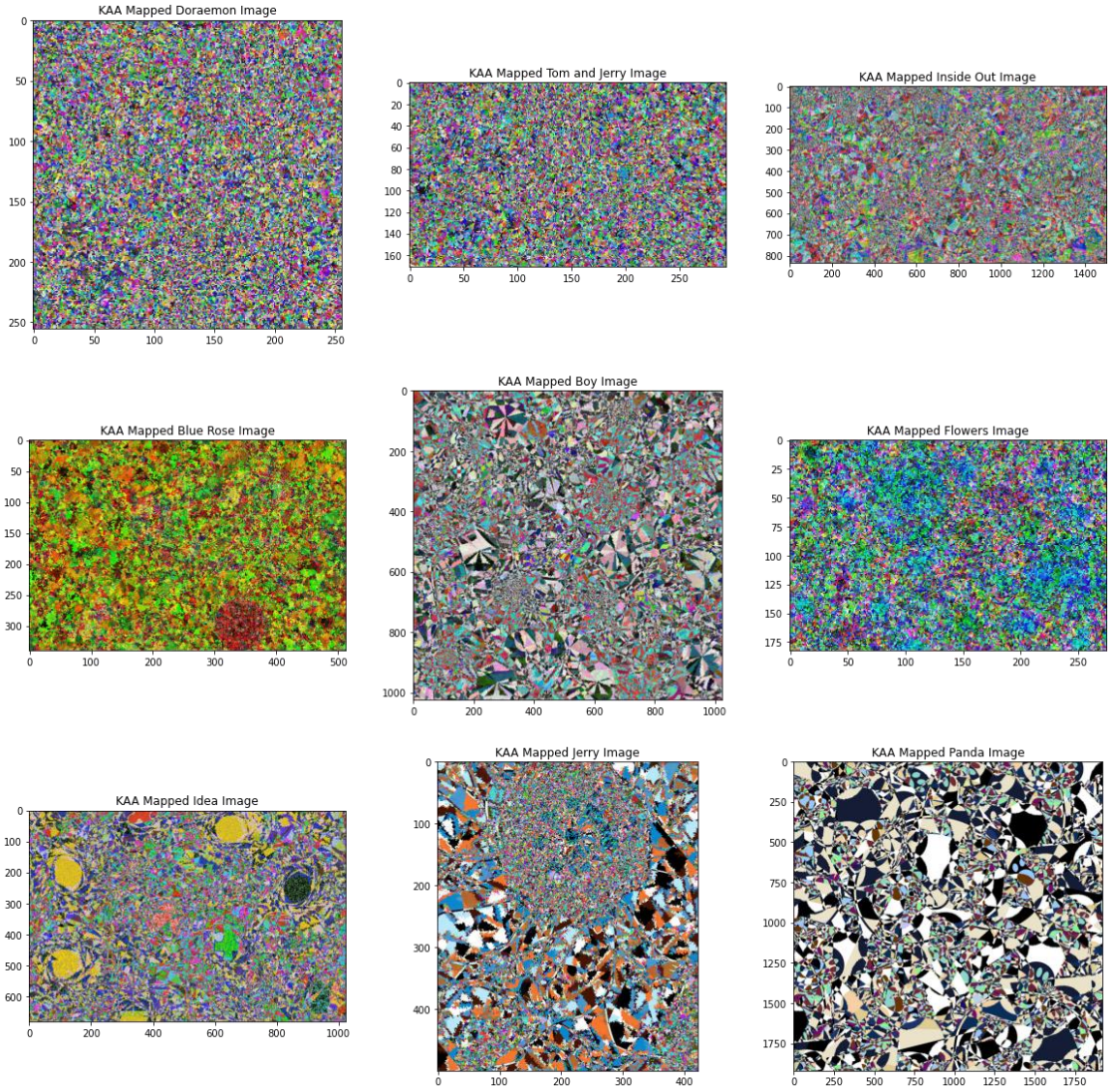



```
# Constants for getting combined transformation transformation indexes
THETA     = 70
VELOCITY1 = 21
VELOCITY2 = 12
TIME      = 34
FREQUENCY = 15

# Fnction to get combined transformation indexes
# This uses Galilean and Rotational transformation for exchanging pixels in t
he image
# x     - the x index to be transformed
# y     - the y index to be transformed
# theta - angle of transformation for rotational transformation
# v1    - velocity for galilean transformation of x index
```

```
# v2    - velocity for galilean transformation of y index
# l     - length for confining the x transformation
# b     - breadth for confining the y transformation
# t     - time for performing galilean transformation
def get_combined_transformation(x, y, theta, v1, v2, l, b, t):
    x1 = x * math.cos(theta) - y * math.sin(theta) + v1 * t
    y1 = x * math.sin(theta) + y * math.cos(theta) + v2 * t
    return int(x1) % l, int(y1) % b

# Function to perform combined gallilean and rotational transformation on an
image
# theta   - angle of transformation for rotational transformation
# v1      - velocity for galilean transformation of x index
# v2      - velocity for galilean transformation of y index
# t       - time for galilean transformation
# f       - frequency of application of diffusion
# image   - image to be transformed
# reverse - whether or not reverse transformation is to be performed, by defa
ult it is false
def perform_combined_transformation_in_image(theta, v1, v2, t, f, image, reve
rse = False):
    x, y, z = np.shape(image)
    blue, green, red = cv2.split(np.array(image))
    row_start, row_end, row_step = 0, x, 1
    col_start, col_end, col_step = 0, y, 1
    if reverse:
        row_start, row_end, row_step = x - 1, -1, -1
        col_start, col_end, col_step = y - 1, -1, -1
    for k in range(f):
        for i in range(row_start, row_end, row_step):
            for j in range(col_start, col_end, col_step):
                new_i, new_j = get_combined_transformation(i, j, theta, v1, v
2, x, y, t)
                red[i][j]   ^= red[new_i][new_j]
                blue[i][j]  ^= blue[new_i][new_j]
                green[i][j] ^= green[new_i][new_j]
    if reverse:
        return cv2.merge([blue, green, red])
    return cv2.merge([red, green, blue])

# Kaa Mapping of images for diffusion of pixels
kaa_mapped_doraemon_image      = perform_combined_transformation_in_image(THE
TA, VELOCITY1, VELOCITY2, TIME, FREQUENCY,
                                                                        rgb
_doraemon_image)
kaa_mapped_tom_and_jerry_image = perform_combined_transformation_in_image(THE
```

```
TA, VELOCITY1, VELOCITY2, TIME, FREQUENCY,
                                                                    rgb
_tom_and_jerry_image)
kaa_mapped_inside_out_image    = perform_combined_transformation_in_image(THE
TA, VELOCITY1, VELOCITY2, TIME, FREQUENCY,
                                                                    rgb
_inside_out_image)
kaa_mapped_blue_rose_image     = perform_combined_transformation_in_image(THE
TA, VELOCITY1, VELOCITY2, TIME, FREQUENCY,
                                                                    rgb
_blue_rose_image)
kaa_mapped_boy_image           = perform_combined_transformation_in_image(THE
TA, VELOCITY1, VELOCITY2, TIME, FREQUENCY,
                                                                    rgb
_boy_image)
kaa_mapped_flowers_image       = perform_combined_transformation_in_image(THE
TA, VELOCITY1, VELOCITY2, TIME, FREQUENCY,
                                                                    rgb
_flowers_image)
kaa_mapped_idea_image          = perform_combined_transformation_in_image(THE
TA, VELOCITY1, VELOCITY2, TIME, FREQUENCY,
                                                                    rgb
_idea_image)
kaa_mapped_jerry_image         = perform_combined_transformation_in_image(THE
TA, VELOCITY1, VELOCITY2, TIME, FREQUENCY,
                                                                    rgb
_jerry_image)
kaa_mapped_panda_image         = perform_combined_transformation_in_image(THE
TA, VELOCITY1, VELOCITY2, TIME, FREQUENCY,
                                                                    rgb
_panda_image)

kaa_mapped_images_list        = [kaa_mapped_doraemon_image, kaa_mapped_tom_an
d_jerry_image, kaa_mapped_inside_out_image,
                                 kaa_mapped_blue_rose_image, kaa_mapped_boy_i
mage, kaa_mapped_flowers_image,
                                 kaa_mapped_idea_image, kaa_mapped_jerry_imag
e, kaa_mapped_panda_image]
kaa_mapped_images_titles_list = ["KAA Mapped Doraemon Image", "KAA Mapped Tom
and Jerry Image", "KAA Mapped Inside Out Image",
                                 "KAA Mapped Blue Rose Image", "KAA Mapped Bo
y Image", "KAA Mapped Flowers Image",
                                 "KAA Mapped Idea Image", "KAA Mapped Jerry I
mage", "KAA Mapped Panda Image"]
```

```
display_images_horizontaly(20, 20, 3, 3, kaa_mapped_images_list, kaa_mapped_i
mages_titles_list)
```




```
# Constants for 2D Logistic Adjusted Sine Map
X = 0.3
Y = 0.3
U = 0.9

# Function to get 2D Logistic Adjusted Sine Map
# This is a combination of Logistic Map and Sine Map to add Chaos to the imag
e
# This is used to create the first key for creating confusion in the image
# x       - The starting value of x for the sequence
# y       - The starting value of y for the sequence
```

```
# u       - The control parameter for the sequence
# limit_x - Number of elements required in the x sequence
# limit_y - Number of elements required in the y sequence
def get_2D_logistic_adjusted_sine_map(x, y, u, limit_x, limit_y):
    def get_next_x(x, y, u):
        return math.sin(math.pi * u * (y + 3) * x * (1 - x))
    def get_next_y(x, y, u):
        return math.sin(math.pi * u * (x + 3) * y * (1 - x))
    result_x = []
    result_y = []
    for i in range(limit_x):
        x = get_next_x(x, y, u)
        result_x.append(x)
    for i in range(limit_y):
        y = get_next_y(x, y, u)
        result_y.append(y)
    return result_x, result_y

# Function to get 2D Logistic Adjusted Sine Map for a particular image
# x     - The starting value of x for the sequence
# y     - The starting value of y for the sequence
# u     - The control parameter for the sequence
# image - The image for which the 2D adjusted logistic sine map sequence is r
equired
def get_2D_logistic_adjusted_sine_map_for_image(x, y, u, image):
    limit_x, limit_y, z = np.shape(image)
    return get_2D_logistic_adjusted_sine_map(x, y, u, limit_x, limit_y)

# The 2D logistic sequences for different images
result_x_doraemon_image,      result_y_doraemon_image      = get_2D_logistic_
adjusted_sine_map_for_image(
    X, Y, U, rgb_doraemon_image)
result_x_tom_and_jerry_image, result_y_tom_and_jerry_image = get_2D_logistic_
adjusted_sine_map_for_image(
    X, Y, U, rgb_tom_and_jerry_image)
result_x_inside_out_image,    result_y_inside_out_image    = get_2D_logistic_
adjusted_sine_map_for_image(
    X, Y, U, rgb_inside_out_image)
result_x_blue_rose_image,     result_y_blue_rose_image     = get_2D_logistic_
adjusted_sine_map_for_image(
    X, Y, U, rgb_blue_rose_image)
result_x_boy_image,           result_y_boy_image           = get_2D_logistic_
adjusted_sine_map_for_image(
    X, Y, U, rgb_boy_image)
result_x_flowers_image,       result_y_flowers_image       = get_2D_logistic_
adjusted_sine_map_for_image(
```

```
    X, Y, U, rgb_flowers_image)
result_x_idea_image,          result_y_idea_image          = get_2D_logistic_
adjusted_sine_map_for_image(
    X, Y, U, rgb_idea_image)
result_x_jerry_image,         result_y_jerry_image         = get_2D_logistic_
adjusted_sine_map_for_image(
    X, Y, U, rgb_jerry_image)
result_x_panda_image,         result_y_panda_image         = get_2D_logistic_
adjusted_sine_map_for_image(
    X, Y, U, rgb_panda_image)

# ### IEEE - 754 double precision convertor

# 1. Convert given number in binary
# 2. Represent converted binary number into scientific notation
# 3. Convert number into IEEE -754 dobule precision 64bit format

# 1st bit - sign (+ve = 0, -ve = 1)
# Next 11 bits - exponent bias + power  (2<sup>(k - 1)</sup> - 1) + p
# Next 52 bits - mantissa

# Function to get exponent of a number in scientific notation
# binary_number - Binary number in string format to get the exponent when the
number is written in scientific format
def get_exponent_in_scientific_notation(binary_number):
    return len([each for each in binary_number]) - 1

# Function to get exponent bias added with the power
# binary_number - Binary number in string format for which the exponent bias
plus power is required
# k             - Number of bits to be considered in the exponent bias plus p
ower, by default this is 11
def get_exponent_bias_plus_power(binary_number, k = 11):
    exponent_bias = 11
    power = get_exponent_in_scientific_notation(binary_number)
    return bin(((2 ** (k - 1)) - 1) + power)[2:]

# Function to convert a binary number to required number of bits
# binary_number - The binary number which needs to be converted to required n
umber of bits
# required_bits - The number of bits to be considered in the binary number
def convert_binary_number_to_required_bits(binary_number, required_bits):
    if len(binary_number) > required_bits:
        return binary_number[:required_bits]
    return '0' * (required_bits - len(binary_number)) + binary_number
```

```
# Function to convert a decimal point number into IEE 754 double precision no
tation
# decimal_point_number - The number in decimal points that need to be convert
ed
def IEEE_754_double_precision_convertor(decimal_point_number):
    sign = '0'
    if decimal_point_number < 0:
        sign = '1'
        decimal_point_number *= -1
    decimal_point_number = '{0:.50f}'.format(decimal_point_number)
    number_before_decimal, number_after_decimal = decimal_point_number.split(
'.')
    number_before_decimal, number_after_decimal = int(number_before_decimal),
int(number_after_decimal)
    binary_number_before_decimal = bin(number_before_decimal)[2:]
    binary_number_after_decimal = bin(number_after_decimal)[2:]
    exponent = get_exponent_bias_plus_power(binary_number_before_decimal)
    mantissa = binary_number_before_decimal[1:] + binary_number_after_decimal
    return sign + convert_binary_number_to_required_bits(exponent, 11) + conv
ert_binary_number_to_required_bits(mantissa, 52)

# The floating point output sequences, x and y,
# resulting from the 2D Sine Logistic Map are then
# converted to 64-bit by utilizing the IEEE-754 double precision conversion.
The control parameter a
# is converted to 64-bit in the same manner. Nevertheless, it is shortened, s
uch that only the first
# 44 bits are employed.

# Converting the 2D Logistic Adjusted Sine Map sequences of different images
in IEEE-754 double precision format
converted_result_x_doraemon_image,      converted_result_y_doraemon_image
= \
[int(IEEE_754_double_precision_convertor(each), 2) for each in result_x_dorae
mon_image], \
[int(IEEE_754_double_precision_convertor(each), 2) for each in result_y_dorae
mon_image]
converted_result_x_tom_and_jerry_image, converted_result_y_tom_and_jerry_imag
e = \
[int(IEEE_754_double_precision_convertor(each), 2) for each in result_x_tom_a
nd_jerry_image],\
[int(IEEE_754_double_precision_convertor(each), 2) for each in result_y_tom_a
nd_jerry_image]
converted_result_x_inside_out_image,    converted_result_y_inside_out_image
= \
[int(IEEE_754_double_precision_convertor(each), 2) for each in result_x_insid
```

```
e_out_image], \
[int(IEEE_754_double_precision_convertor(each), 2) for each in result_y_insid
e_out_image]
converted_result_x_blue_rose_image,     converted_result_y_blue_rose_image
= \
[int(IEEE_754_double_precision_convertor(each), 2) for each in result_x_blue_
rose_image], \
[int(IEEE_754_double_precision_convertor(each), 2) for each in result_y_blue_
rose_image]
converted_result_x_boy_image,           converted_result_y_boy_image
= \
[int(IEEE_754_double_precision_convertor(each), 2) for each in result_x_boy_i
mage], \
[int(IEEE_754_double_precision_convertor(each), 2) for each in result_y_boy_i
mage]
converted_result_x_flowers_image,       converted_result_y_flowers_image
= \
[int(IEEE_754_double_precision_convertor(each), 2) for each in result_x_flowe
rs_image], \
[int(IEEE_754_double_precision_convertor(each), 2) for each in result_y_flowe
rs_image]
converted_result_x_idea_image,          converted_result_y_idea_image
= \
[int(IEEE_754_double_precision_convertor(each), 2) for each in result_x_idea_
image], \
[int(IEEE_754_double_precision_convertor(each), 2) for each in result_y_idea_
image]
converted_result_x_jerry_image,         converted_result_y_jerry_image
= \
[int(IEEE_754_double_precision_convertor(each), 2) for each in result_x_jerry
_image], \
[int(IEEE_754_double_precision_convertor(each), 2) for each in result_y_jerry
_image]
converted_result_x_panda_image,         converted_result_y_panda_image
= \
[int(IEEE_754_double_precision_convertor(each), 2) for each in result_x_panda
_image], \
[int(IEEE_754_double_precision_convertor(each), 2) for each in result_y_panda
_image]

# Converting the control parameter into the IEEE 754 double precision format
control_a = int(convert_binary_number_to_required_bits(IEEE_754_double_precis
ion_convertor(U), 44), 2)
```

```
# Constants for Linear Congruential Generator
L10, AL1, C1, M1 = 358644446, 27, 0, (2 ** 52) - 1
L20, AL2, C2, M2 = 2700000, 37, 1, (2 ** 32) - 1

# Function to calculate Linear Congruential Sequence
# x     - The starting of the sequence
# a     - The control paramer of the sequence
# c     - The constant of the sequence
# m     - The max limit to confine the output
# limit - The number of elements to be generated in the sequence
def linear_congruential_generator(x, a, c, m, limit):
    def get_next_element(x, a, c, m):
        return (a * x - 1 + c) % m
    result = []
    for i in range(limit):
        result.append(x)
        x = get_next_element(x, a, c, m)
    return result

# Function to get the linear congruential sequence for length and breadth of
the image
# x          - The starting of the sequence
# a          - The control paramer of the sequence
# c          - The constant of the sequence
# m          - The max limit to confine the output
# image      - The image for which the linear congruential sequence is requir
ed
# is_length  - Whether the sequence is required according to the length of th
e image
# is_breadth - Whether the sequence is required according to the breadth of t
he image
def get_linear_congruential_sequence_for_image(x, a, c, m, image, is_length,
is_breadth):
    limit_x, limit_y, z = np.shape(image)
    if is_length:
        return linear_congruential_generator(x, a, c, m, limit_x)
    if is_breadth:
        return linear_congruential_generator(x, a, c, m, limit_y)
    return []

# The 2D logistic sequences for different images
result_1_doraemon_image,      result_2_doraemon_image      = \
get_linear_congruential_sequence_for_image(L10, AL1, C1, M1, rgb_doraemon_ima
ge, True, False), \
get_linear_congruential_sequence_for_image(L20, AL2, C2, M2, rgb_doraemon_ima
ge, False, True)
result_1_tom_and_jerry_image, result_2_tom_and_jerry_image = \
```

```
get_linear_congruential_sequence_for_image(L10, AL1, C1, M1, rgb_tom_and_jerr
y_image, True, False), \
get_linear_congruential_sequence_for_image(L20, AL2, C2, M2, rgb_tom_and_jerr
y_image, False, True)
result_1_inside_out_image,    result_2_inside_out_image    = \
get_linear_congruential_sequence_for_image(L10, AL1, C1, M1, rgb_inside_out_i
mage, True, False), \
get_linear_congruential_sequence_for_image(L20, AL2, C2, M2, rgb_inside_out_i
mage, False, True)
result_1_blue_rose_image,     result_2_blue_rose_image     = \
get_linear_congruential_sequence_for_image(L10, AL1, C1, M1, rgb_blue_rose_im
age, True, False), \
get_linear_congruential_sequence_for_image(L20, AL2, C2, M2, rgb_blue_rose_im
age, False, True)
result_1_boy_image,           result_2_boy_image           = \
get_linear_congruential_sequence_for_image(L10, AL1, C1, M1, rgb_boy_image, T
rue, False), \
get_linear_congruential_sequence_for_image(L20, AL2, C2, M2, rgb_boy_image, F
alse, True)
result_1_flowers_image,       result_2_flowers_image       = \
get_linear_congruential_sequence_for_image(L10, AL1, C1, M1, rgb_flowers_imag
e, True, False), \
get_linear_congruential_sequence_for_image(L20, AL2, C2, M2, rgb_flowers_imag
e, False, True)
result_1_idea_image,          result_2_idea_image          = \
get_linear_congruential_sequence_for_image(L10, AL1, C1, M1, rgb_idea_image,
True, False), \
get_linear_congruential_sequence_for_image(L20, AL2, C2, M2, rgb_idea_image,
False, True)
result_1_jerry_image,         result_2_jerry_image         = \
get_linear_congruential_sequence_for_image(L10, AL1, C1, M1, rgb_jerry_image,
True, False), \
get_linear_congruential_sequence_for_image(L20, AL2, C2, M2, rgb_jerry_image,
False, True)
result_1_panda_image,         result_2_panda_image         = \
get_linear_congruential_sequence_for_image(L10, AL1, C1, M1, rgb_panda_image,
True, False), \
get_linear_congruential_sequence_for_image(L20, AL2, C2, M2, rgb_panda_image,
False, True)

# Converting the Linear Congruential Sequences of different images in IEEE-75
4 double precision format
converted_result_1_doraemon_image,      converted_result_2_doraemon_image
= \
[int(convert_binary_number_to_required_bits(IEEE_754_double_precision_convert
or(each), 52), 2) \
```

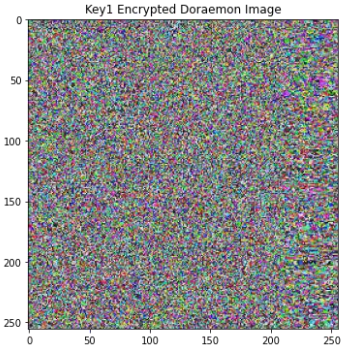
```
 for each in result_1_doraemon_image], \
[int(convert_binary_number_to_required_bits(IEEE_754_double_precision_convert
or(each), 32), 2) \
 for each in result_2_doraemon_image]
converted_result_1_tom_and_jerry_image, converted_result_2_tom_and_jerry_imag
e = \
[int(convert_binary_number_to_required_bits(IEEE_754_double_precision_convert
or(each), 52), 2) \
 for each in result_1_tom_and_jerry_image],\
[int(convert_binary_number_to_required_bits(IEEE_754_double_precision_convert
or(each), 32), 2) \
 for each in result_2_tom_and_jerry_image]
converted_result_1_inside_out_image,    converted_result_2_inside_out_image
= \
[int(convert_binary_number_to_required_bits(IEEE_754_double_precision_convert
or(each), 52), 2) \
 for each in result_1_inside_out_image], \
[int(convert_binary_number_to_required_bits(IEEE_754_double_precision_convert
or(each), 32), 2) \
 for each in result_2_inside_out_image]
converted_result_1_blue_rose_image,     converted_result_2_blue_rose_image
= \
[int(convert_binary_number_to_required_bits(IEEE_754_double_precision_convert
or(each), 52), 2) \
 for each in result_1_blue_rose_image], \
[int(convert_binary_number_to_required_bits(IEEE_754_double_precision_convert
or(each), 32), 2) \
 for each in result_2_blue_rose_image]
converted_result_1_boy_image,           converted_result_2_boy_image
= \
[int(convert_binary_number_to_required_bits(IEEE_754_double_precision_convert
or(each), 52), 2) \
 for each in result_1_boy_image], \
[int(convert_binary_number_to_required_bits(IEEE_754_double_precision_convert
or(each), 32), 2) \
 for each in result_2_boy_image]
converted_result_1_flowers_image,       converted_result_2_flowers_image
= \
[int(convert_binary_number_to_required_bits(IEEE_754_double_precision_convert
or(each), 52), 2) \
 for each in result_1_flowers_image], \
[int(convert_binary_number_to_required_bits(IEEE_754_double_precision_convert
or(each), 32), 2) \
 for each in result_2_flowers_image]
converted_result_1_idea_image,          converted_result_2_idea_image
= \
```

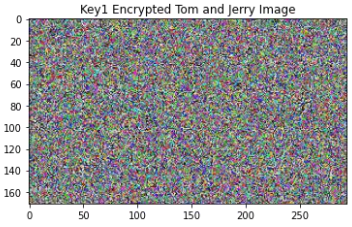
```
[int(convert_binary_number_to_required_bits(IEEE_754_double_precision_convert
or(each), 52), 2) \
 for each in result_1_idea_image], \
[int(convert_binary_number_to_required_bits(IEEE_754_double_precision_convert
or(each), 32), 2) \
 for each in result_2_idea_image]
converted_result_1_jerry_image,         converted_result_2_jerry_image
= \
[int(convert_binary_number_to_required_bits(IEEE_754_double_precision_convert
or(each), 52), 2) \
 for each in result_1_jerry_image], \
[int(convert_binary_number_to_required_bits(IEEE_754_double_precision_convert
or(each), 32), 2) \
 for each in result_2_jerry_image]
converted_result_1_panda_image,         converted_result_2_panda_image
= \
[int(convert_binary_number_to_required_bits(IEEE_754_double_precision_convert
or(each), 52), 2) \
 for each in result_1_panda_image], \
[int(convert_binary_number_to_required_bits(IEEE_754_double_precision_convert
or(each), 32), 2) \
 for each in result_2_panda_image]

# Function to generate first key to create confusion for a particular image
# image    - Image for which the first key has to be generated
# result_x - first result obtained from 2D Logisctic Adjusted Sine Map
# result_y - second result obtained from 2D Logistic Adjusted Sine Map
# result_1 - first result obtained from Linear Congruential Generator
# result_2 - second result obtained from Linear Congruential Generator
def generate_key1_for_an_image(image, result_x, result_y, control_a, result_1
, result_2):
    limit_x, limit_y, z = np.shape(image)
    return [[(result_x[i] + result_y[j] + control_a + result_1[i] + result_2[
j]) % 256 \
             for j in range(limit_y)] for i in range(limit_x)]

# Getting the first keys for different images
key1_doraemon_image      = generate_key1_for_an_image(rgb_doraemon_image,
                                                      converted_result_x_doraemon_
image,
                                                      converted_result_y_doraemon_
image,
                                                      control_a,
                                                      converted_result_1_doraemon_
image,
                                                      converted_result_2_doraemon_
```

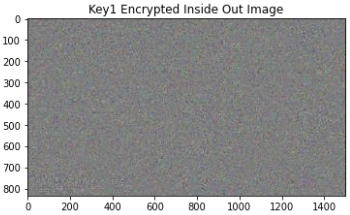
```
image)
key1_tom_and_jerry_image = generate_key1_for_an_image(rgb_tom_and_jerry_image
,
                                                       converted_result_x_tom_and_j
erry_image,
                                                       converted_result_y_tom_and_j
erry_image,
                                                       control_a,
                                                       converted_result_1_tom_and_j
erry_image,
                                                       converted_result_2_tom_and_j
erry_image)
key1_inside_out_image    = generate_key1_for_an_image(rgb_inside_out_image,
                                                       converted_result_x_inside_ou
t_image,
                                                       converted_result_y_inside_ou
t_image,
                                                       control_a,
                                                       converted_result_1_inside_ou
t_image,
                                                       converted_result_2_inside_ou
t_image)
key1_blue_rose_image     = generate_key1_for_an_image(rgb_blue_rose_image,
                                                       converted_result_x_blue_rose
_image,
                                                       converted_result_y_blue_rose
_image,
                                                       control_a,
                                                       converted_result_1_blue_rose
_image,
                                                       converted_result_2_blue_rose
_image)
key1_boy_image = generate_key1_for_an_image(rgb_boy_image,
                                                       converted_result_x_boy_image
,
                                                       converted_result_y_boy_image
,
                                                       control_a,
                                                       converted_result_1_boy_image
,
                                                       converted_result_2_boy_image
)
key1_flowers_image = generate_key1_for_an_image(rgb_flowers_image,
                                                       converted_result_x_flowers_i
mage,
                                                       converted_result_y_flowers_i
```

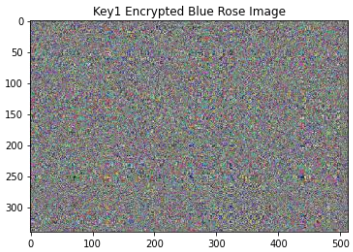
```
mage,
                                                    control_a,
                                                    converted_result_1_flowers_i
mage,
                                                    converted_result_2_flowers_i
mage)
key1_idea_image = generate_key1_for_an_image(rgb_idea_image,
                                                    converted_result_x_idea_imag
e,
                                                    converted_result_y_idea_imag
e,
                                                    control_a,
                                                    converted_result_1_idea_imag
e,
                                                    converted_result_2_idea_imag
e)
key1_jerry_image = generate_key1_for_an_image(rgb_jerry_image,
                                                    converted_result_x_jerry_ima
ge,
                                                    converted_result_y_jerry_ima
ge,
                                                    control_a,
                                                    converted_result_1_jerry_ima
ge,
                                                    converted_result_2_jerry_ima
ge)
key1_panda_image = generate_key1_for_an_image(rgb_panda_image,
                                                    converted_result_x_panda_ima
ge,
                                                    converted_result_y_panda_ima
ge,
                                                    control_a,
                                                    converted_result_1_panda_ima
ge,
                                                    converted_result_2_panda_ima
ge)

# Function to get key masked image
# image - Image to which the key encryption has to be applied
# key   - Key for encryption
def get_key_encrypted_image(image, key):
    limit_x, limit_y, z = np.shape(image)
    blue, green, red = cv2.split(np.array(image))
    for i in range(limit_x):
        for j in range(limit_y):
            red[i][j]   ^= key[i][j]
```

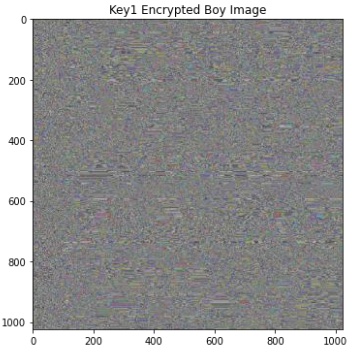
```
            blue[i][j]  ^= key[i][j]
            green[i][j] ^= key[i][j]
    return cv2.merge([red, green, blue])

# Key 1 Encrypted images
key1_encrypted_doraemon_image      = get_key_encrypted_image(kaa_mapped_dorae
mon_image, key1_doraemon_image)
key1_encrypted_tom_and_jerry_image = get_key_encrypted_image(kaa_mapped_tom_a
nd_jerry_image, key1_tom_and_jerry_image)
key1_encrypted_inside_out_image    = get_key_encrypted_image(kaa_mapped_insid
e_out_image, key1_inside_out_image)
key1_encrypted_blue_rose_image     = get_key_encrypted_image(kaa_mapped_blue_
rose_image, key1_blue_rose_image)
key1_encrypted_boy_image           = get_key_encrypted_image(kaa_mapped_boy_i
mage, key1_boy_image)
key1_encrypted_flowers_image       = get_key_encrypted_image(kaa_mapped_flowe
rs_image, key1_flowers_image)
key1_encrypted_idea_image          = get_key_encrypted_image(kaa_mapped_idea_
image, key1_idea_image)
key1_encrypted_jerry_image         = get_key_encrypted_image(kaa_mapped_jerry
_image, key1_jerry_image)
key1_encrypted_panda_image         = get_key_encrypted_image(kaa_mapped_panda
_image, key1_panda_image)

key1_encrypted_images_list        = [key1_encrypted_doraemon_image, key1_encr
ypted_tom_and_jerry_image,
                                     key1_encrypted_inside_out_image, key1_en
crypted_blue_rose_image,
                                     key1_encrypted_boy_image, key1_encrypted
_flowers_image,
                                     key1_encrypted_idea_image, key1_encrypte
d_jerry_image, key1_encrypted_panda_image]
key1_encrypted_images_titles_list = ["Key1 Encrypted Doraemon Image", "Key1 E
ncrypted Tom and Jerry Image",
                                     "Key1 Encrypted Inside Out Image", "Key1
Encrypted Blue Rose Image",
                                     "Key1 Encrypted Boy Image", "Key1 Encryp
ted Flowers Image",
                                     "Key1 Encrypted Idea Image", "Key1 Encry
pted Jerry Image", "Key1 Encrypted Panda Image"]

display_images_horizontaly(20, 20, 3, 3, key1_encrypted_images_list, key1_enc
rypted_images_titles_list)
```

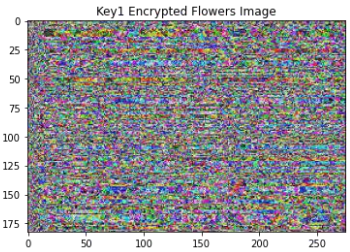


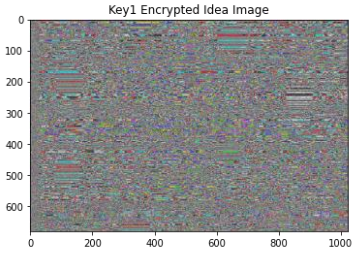


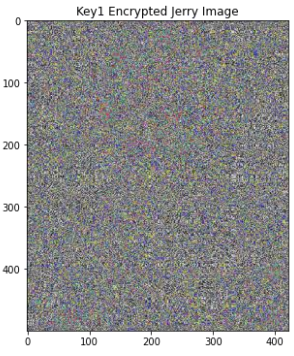


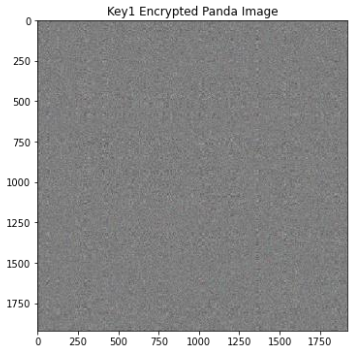













```
# Tent Map Sequence Constants
X_TENT_MAP = 0.5
C_TENT_MAP = 1.5

# Function to get Tent Map sequence for generation of second key
# x     - The begining of tent map sequence
# c     - The control parameter
# limit - The number of elements required in the sequence
def get_tent_map(x, c, limit):
    def get_next_element(x, c):
        if 0 <= x <= 0.5:
            return c * x
        if 0.5 <= x <= 1:
            return c * (1 - x)
```

```
        return x
    result = []
    for i in range(limit):
        result.append(x)
        x = get_next_element(x, c)
    return result

# Function to get Tent Map Sequence for a given Image
def get_tent_map_sequence_for_an_image(image, x, c):
    limit_x, limit_y, z = np.shape(image)
    return get_tent_map(x, c, limit_x)

# Tent map sequences for the different images
tent_map_result_doraemon_image      = get_tent_map_sequence_for_an_image(rgb_
doraemon_image, X_TENT_MAP, C_TENT_MAP)
tent_map_result_tom_and_jerry_image = get_tent_map_sequence_for_an_image(rgb_
tom_and_jerry_image, X_TENT_MAP, C_TENT_MAP)
tent_map_result_inside_out_image    = get_tent_map_sequence_for_an_image(rgb_
inside_out_image, X_TENT_MAP, C_TENT_MAP)
tent_map_result_blue_rose_image     = get_tent_map_sequence_for_an_image(rgb_
blue_rose_image, X_TENT_MAP, C_TENT_MAP)
tent_map_result_boy_image           = get_tent_map_sequence_for_an_image(rgb_
boy_image, X_TENT_MAP, C_TENT_MAP)
tent_map_result_flowers_image       = get_tent_map_sequence_for_an_image(rgb_
flowers_image, X_TENT_MAP, C_TENT_MAP)
tent_map_result_idea_image          = get_tent_map_sequence_for_an_image(rgb_
idea_image, X_TENT_MAP, C_TENT_MAP)
tent_map_result_jerry_image         = get_tent_map_sequence_for_an_image(rgb_
jerry_image, X_TENT_MAP, C_TENT_MAP)
tent_map_result_panda_image         = get_tent_map_sequence_for_an_image(rgb_
panda_image, X_TENT_MAP, C_TENT_MAP)

# IEEE 754 double precision converted tent map sequences for images
converted_tent_map_result_doraemon_image      = [int(convert_binary_number_to
_required_bits(
    IEEE_754_double_precision_convertor(each), 327), 2) for each in tent_map_
result_doraemon_image]
converted_tent_map_result_tom_and_jerry_image = [int(convert_binary_number_to
_required_bits(
    IEEE_754_double_precision_convertor(each), 327), 2) for each in tent_map_
result_tom_and_jerry_image]
converted_tent_map_result_inside_out_image    = [int(convert_binary_number_to
_required_bits(
    IEEE_754_double_precision_convertor(each), 327), 2) for each in tent_map_
result_inside_out_image]
converted_tent_map_result_blue_rose_image     = [int(convert_binary_number_to
_required_bits(
```

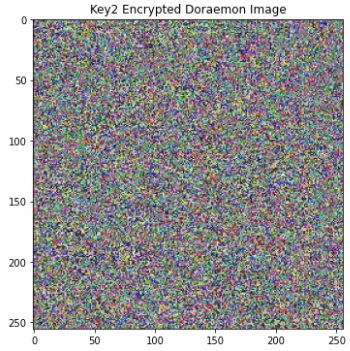
```
    IEEE_754_double_precision_convertor(each), 327), 2) for each in tent_map_
result_blue_rose_image ]
converted_tent_map_result_boy_image           = [int(convert_binary_number_to
_required_bits(
    IEEE_754_double_precision_convertor(each), 327), 2) for each in tent_map_
result_boy_image]
converted_tent_map_result_flowers_image       = [int(convert_binary_number_to
_required_bits(
    IEEE_754_double_precision_convertor(each), 327), 2) for each in tent_map_
result_flowers_image]
converted_tent_map_result_idea_image          = [int(convert_binary_number_to
_required_bits(
    IEEE_754_double_precision_convertor(each), 327), 2) for each in tent_map_
result_idea_image]
converted_tent_map_result_jerry_image         = [int(convert_binary_number_to
_required_bits(
    IEEE_754_double_precision_convertor(each), 327), 2) for each in tent_map_
result_jerry_image]
converted_tent_map_result_panda_image         = [int(convert_binary_number_to
_required_bits(
    IEEE_754_double_precision_convertor(each), 327), 2) for each in tent_map_
result_panda_image]

# Bernouilli Map Constants
X_BERNOULLI_MAP = (0.25 * 1.75) - 0.5
A_BERNOULLI_MAP = 0.5
B_BERNOULLI_MAP = 1.75

# Function to get the bernoulli map sequence for the given image
# x     - The begining of the sequence for the bernoulli map
# a     - The control parameter for the sequence
# b     - The constant parameter for the sequence
# limit - The number of elements to be generated for the sequence
def get_bernoulli_map(x, a, b, limit):
    def get_next_element(x, a, b):
        if -a <= x <= 0:
            return (b * x) - a
        elif 0 <= x <= a:
            return (b * x) + a
        return 0
    result = []
    for i in range(limit):
        result.append(x)
        x = get_next_element(x, a, b)
    return result
```

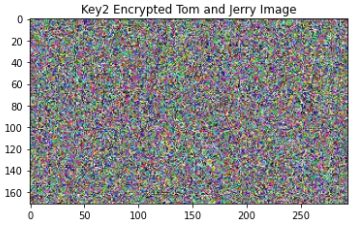
```
def get_bernoulli_map_sequence_for_an_image(image, x, a, b):
    limit_x, limit_y, z = np.shape(image)
    return get_bernoulli_map(x, a, b, limit_y)

# Bernoulli map sequences for the different images
bernoulli_map_result_doraemon_image      = get_bernoulli_map_sequence_for_an_
image(
    rgb_doraemon_image, X_BERNOULLI_MAP, A_BERNOULLI_MAP, B_BERNOULLI_MAP)
bernoulli_map_result_tom_and_jerry_image = get_bernoulli_map_sequence_for_an_
image(
    rgb_tom_and_jerry_image, X_BERNOULLI_MAP, A_BERNOULLI_MAP, B_BERNOULLI_MA
P)
bernoulli_map_result_inside_out_image    = get_bernoulli_map_sequence_for_an_
image(
    rgb_inside_out_image, X_BERNOULLI_MAP, A_BERNOULLI_MAP, B_BERNOULLI_MAP)
bernoulli_map_result_blue_rose_image     = get_bernoulli_map_sequence_for_an_
image(
    rgb_blue_rose_image, X_BERNOULLI_MAP, A_BERNOULLI_MAP, B_BERNOULLI_MAP)
bernoulli_map_result_boy_image           = get_bernoulli_map_sequence_for_an_
image(
    rgb_boy_image, X_BERNOULLI_MAP, A_BERNOULLI_MAP, B_BERNOULLI_MAP)
bernoulli_map_result_flowers_image       = get_bernoulli_map_sequence_for_an_
image(
    rgb_flowers_image, X_BERNOULLI_MAP, A_BERNOULLI_MAP, B_BERNOULLI_MAP)
bernoulli_map_result_idea_image          = get_bernoulli_map_sequence_for_an_
image(
    rgb_idea_image, X_BERNOULLI_MAP, A_BERNOULLI_MAP, B_BERNOULLI_MAP)
bernoulli_map_result_jerry_image         = get_bernoulli_map_sequence_for_an_
image(
    rgb_jerry_image, X_BERNOULLI_MAP, A_BERNOULLI_MAP, B_BERNOULLI_MAP)
bernoulli_map_result_panda_image         = get_bernoulli_map_sequence_for_an_
image(
    rgb_panda_image, X_BERNOULLI_MAP, A_BERNOULLI_MAP, B_BERNOULLI_MAP)

# IEEE 754 double precision converted tent map sequences for images
converted_bernoulli_map_result_doraemon_image      = [int(convert_binary_numb
er_to_required_bits(
    IEEE_754_double_precision_convertor(each), 327), 2) for each in bernoulli
_map_result_doraemon_image]
converted_bernoulli_map_result_tom_and_jerry_image = [int(convert_binary_numb
er_to_required_bits(
    IEEE_754_double_precision_convertor(each), 327), 2) for each in bernoulli
_map_result_tom_and_jerry_image]
converted_bernoulli_map_result_inside_out_image    = [int(convert_binary_numb
er_to_required_bits(
    IEEE_754_double_precision_convertor(each), 327), 2) for each in bernoulli
```

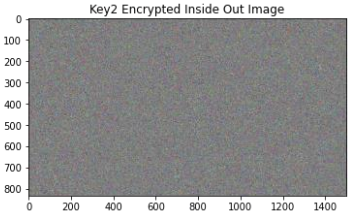
```
_map_result_inside_out_image]
converted_bernoulli_map_result_blue_rose_image     = [int(convert_binary_numb
er_to_required_bits(
    IEEE_754_double_precision_convertor(each), 327), 2) for each in bernoulli
_map_result_blue_rose_image ]
converted_bernoulli_map_result_boy_image           = [int(convert_binary_numb
er_to_required_bits(
    IEEE_754_double_precision_convertor(each), 327), 2) for each in bernoulli
_map_result_boy_image]
converted_bernoulli_map_result_flowers_image       = [int(convert_binary_numb
er_to_required_bits(
    IEEE_754_double_precision_convertor(each), 327), 2) for each in bernoulli
_map_result_flowers_image]
converted_bernoulli_map_result_idea_image          = [int(convert_binary_numb
er_to_required_bits(
    IEEE_754_double_precision_convertor(each), 327), 2) for each in bernoulli
_map_result_idea_image]
converted_bernoulli_map_result_jerry_image         = [int(convert_binary_numb
er_to_required_bits(
    IEEE_754_double_precision_convertor(each), 327), 2) for each in bernoulli
_map_result_jerry_image]
converted_bernoulli_map_result_panda_image         = [int(convert_binary_numb
er_to_required_bits(
    IEEE_754_double_precision_convertor(each), 327), 2) for each in bernoulli
_map_result_panda_image]

# Function to generate second encryption key from tent map and bernoulli map
sequences
# tent_map_result      - Sequence generated using the tent map equation
# bernoulli_map_result - Sequence generated using the bernoulli map equation
# image                - The image for which the key is required
def generate_key2_for_an_image(tent_map_result, bernoulli_map_result, image):
    limit_x, limit_y, x = np.shape(image)
    return [[int(tent_map_result[i] + bernoulli_map_result[j]) % 256 for j in
range(limit_y)] for i in range(limit_x)]

# Second keys for creating another level of confusion in the images
key2_doraemon_image      = generate_key2_for_an_image(converted_tent_map_resu
lt_doraemon_image,
                                                      converted_bernoulli_map_res
ult_doraemon_image,
                                                      rgb_doraemon_image)
key2_tom_and_jerry_image = generate_key2_for_an_image(converted_tent_map_resu
lt_tom_and_jerry_image,
                                                      converted_bernoulli_map_res
ult_tom_and_jerry_image,
```

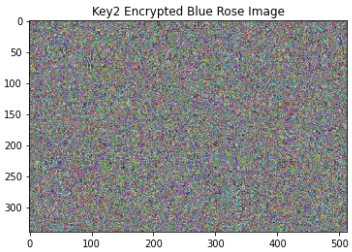
```
                                                    rgb_tom_and_jerry_image)
key2_inside_out_image    = generate_key2_for_an_image(converted_tent_map_resu
lt_inside_out_image,
                                                    converted_bernoulli_map_res
ult_inside_out_image,
                                                    rgb_inside_out_image)
key2_blue_rose_image     = generate_key2_for_an_image(converted_tent_map_resu
lt_blue_rose_image,
                                                    converted_bernoulli_map_res
ult_blue_rose_image,
                                                    rgb_blue_rose_image)
key2_boy_image           = generate_key2_for_an_image(converted_tent_map_resu
lt_boy_image,
                                                    converted_bernoulli_map_res
ult_boy_image,
                                                    rgb_boy_image)
key2_flowers_image       = generate_key2_for_an_image(converted_tent_map_resu
lt_flowers_image,
                                                    converted_bernoulli_map_res
ult_flowers_image,
                                                    rgb_flowers_image)
key2_idea_image          = generate_key2_for_an_image(converted_tent_map_resu
lt_idea_image,
                                                    converted_bernoulli_map_res
ult_idea_image,
                                                    rgb_idea_image)
key2_jerry_image         = generate_key2_for_an_image(converted_tent_map_resu
lt_jerry_image,
                                                    converted_bernoulli_map_res
ult_jerry_image,
                                                    rgb_jerry_image)
key2_panda_image         = generate_key2_for_an_image(converted_tent_map_resu
lt_panda_image,
                                                    converted_bernoulli_map_res
ult_panda_image,
                                                    rgb_panda_image)

# Key 2 Encrypted images
key2_encrypted_doraemon_image      = get_key_encrypted_image(key1_encrypted_d
oraemon_image, key2_doraemon_image)
key2_encrypted_tom_and_jerry_image = get_key_encrypted_image(key1_encrypted_t
om_and_jerry_image, key2_tom_and_jerry_image)
key2_encrypted_inside_out_image    = get_key_encrypted_image(key1_encrypted_i
nside_out_image, key2_inside_out_image)
key2_encrypted_blue_rose_image     = get_key_encrypted_image(key1_encrypted_b
lue_rose_image, key2_blue_rose_image)
```

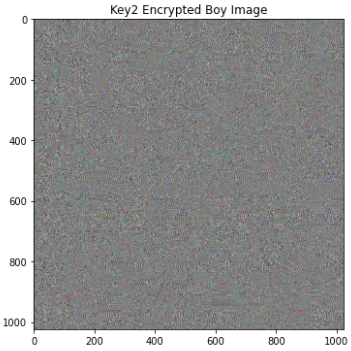
```
key2_encrypted_boy_image           = get_key_encrypted_image(key1_encrypted_b
oy_image, key2_boy_image)
key2_encrypted_flowers_image       = get_key_encrypted_image(key1_encrypted_f
lowers_image, key2_flowers_image)
key2_encrypted_idea_image          = get_key_encrypted_image(key1_encrypted_i
dea_image, key2_idea_image)
key2_encrypted_jerry_image         = get_key_encrypted_image(key1_encrypted_j
erry_image, key2_jerry_image)
key2_encrypted_panda_image         = get_key_encrypted_image(key1_encrypted_p
anda_image, key2_panda_image)

key2_encrypted_images_list        = [key2_encrypted_doraemon_image, key2_encr
ypted_tom_and_jerry_image,
                                     key2_encrypted_inside_out_image, key2_en
crypted_blue_rose_image,
                                     key2_encrypted_boy_image, key2_encrypted
_flowers_image,
                                     key2_encrypted_idea_image, key2_encrypte
d_jerry_image, key2_encrypted_panda_image]
key2_encrypted_images_titles_list = ["Key2 Encrypted Doraemon Image", "Key2 E
ncrypted Tom and Jerry Image",
                                     "Key2 Encrypted Inside Out Image", "Key2
Encrypted Blue Rose Image",
                                     "Key2 Encrypted Boy Image", "Key2 Encryp
ted Flowers Image",
                                     "Key2 Encrypted Idea Image", "Key2 Encry
pted Jerry Image", "Key2 Encrypted Panda Image"]

display_images_horizontaly(20, 20, 3, 3, key2_encrypted_images_list, key2_enc
rypted_images_titles_list)
```

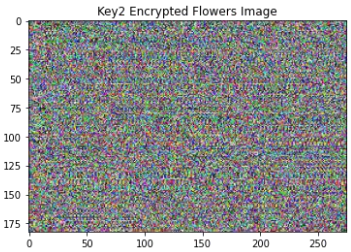


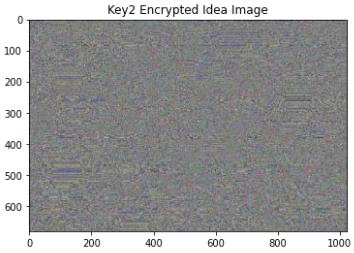


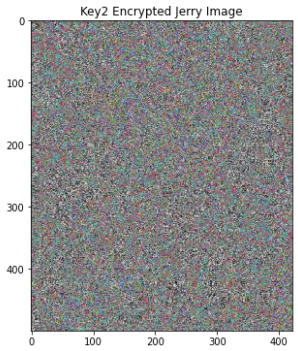


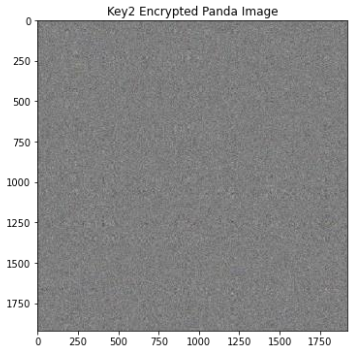








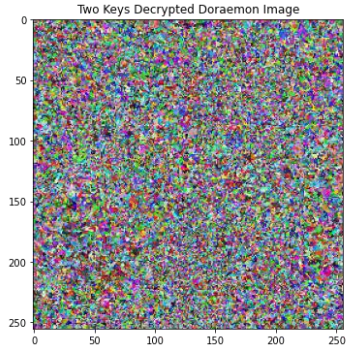


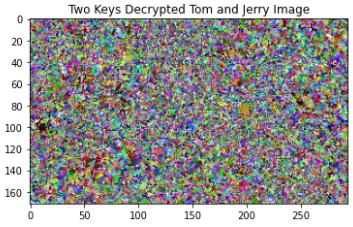


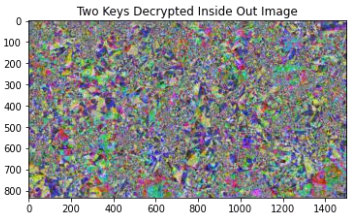

```
# Splitting the different ecncrypted images into red, blue and green images
blue_encrypted_doraemon_image,      green_encrypted_doraemon_image,      red_
encrypted_doraemon_image      = \
cv2.split(np.array(key2_encrypted_doraemon_image))
blue_encrypted_tom_and_jerry_image, green_encrypted_tom_and_jerry_image, red_
encrypted_tom_and_jerry_image = \
cv2.split(np.array(key2_encrypted_tom_and_jerry_image))
blue_encrypted_inside_out_image,    green_encrypted_inside_out_image,    red_
encrypted_inside_out_image    = \
cv2.split(np.array(key2_encrypted_inside_out_image))
blue_encrypted_blue_rose_image,     green_encrypted_blue_rose_image,     red_
encrypted_blue_rose_image     = \
cv2.split(np.array(key2_encrypted_blue_rose_image))
```

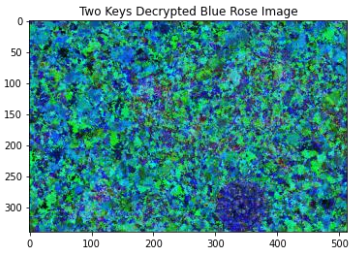
```
blue_encrypted_boy_image,           green_encrypted_boy_image,           red_
encrypted_boy_image           = \
cv2.split(np.array(key2_encrypted_boy_image))
blue_encrypted_flowers_image,       green_encrypted_flowers_image,       red_
encrypted_flowers_image       = \
cv2.split(np.array(key2_encrypted_flowers_image))
blue_encrypted_idea_image,          green_encrypted_idea_image,          red_
encrypted_idea_image          = \
cv2.split(np.array(key2_encrypted_idea_image))
blue_encrypted_jerry_image,         green_encrypted_jerry_image,         red_
encrypted_jerry_image         = \
cv2.split(np.array(key2_encrypted_jerry_image))
blue_encrypted_panda_image,         green_encrypted_panda_image,         red_
encrypted_panda_image         = \
cv2.split(np.array(key2_encrypted_panda_image))

# Function to decrypt image encrypted using two keys, key1 and key2
# key1  - First key used for image encryption
# key2  - Second key used for image encryption
# image - The Image to decrypt
def decrypt_two_keys_encrypted_image(key1, key2, image):
    limit_x, limit_y, z = np.shape(image)
    blue, green, red = cv2.split(np.array(image))
    for i in range(limit_x):
        for j in range(limit_y):
            red[i][j]   ^= (key1[i][j] ^ key2[i][j])
            blue[i][j]  ^= (key1[i][j] ^ key2[i][j])
            green[i][j] ^= (key1[i][j] ^ key2[i][j])
    return cv2.merge([red, green, blue])

two_keys_decrypted_doraemon_image      = decrypt_two_keys_encrypted_image(
    key1_doraemon_image, key2_doraemon_image, key2_encrypted_doraemon_image)
two_keys_decrypted_tom_and_jerry_image = decrypt_two_keys_encrypted_image(
    key1_tom_and_jerry_image, key2_tom_and_jerry_image, key2_encrypted_tom_an
d_jerry_image)
two_keys_decrypted_inside_out_image    = decrypt_two_keys_encrypted_image(
    key1_inside_out_image, key2_inside_out_image, key2_encrypted_inside_out_i
mage)
two_keys_decrypted_blue_rose_image     = decrypt_two_keys_encrypted_image(
    key1_blue_rose_image, key2_blue_rose_image, key2_encrypted_blue_rose_imag
e)
two_keys_decrypted_boy_image           = decrypt_two_keys_encrypted_image(
    key1_boy_image, key2_boy_image, key2_encrypted_boy_image)
two_keys_decrypted_flowers_image       = decrypt_two_keys_encrypted_image(
    key1_flowers_image, key2_flowers_image, key2_encrypted_flowers_image)
two_keys_decrypted_idea_image          = decrypt_two_keys_encrypted_image(
```

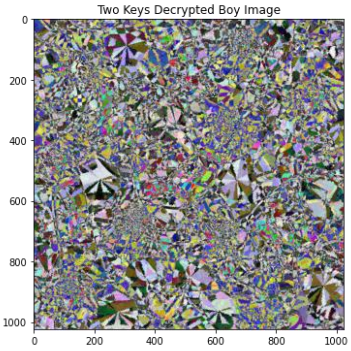
```
    key1_idea_image, key2_idea_image, key2_encrypted_idea_image)
two_keys_decrypted_jerry_image          = decrypt_two_keys_encrypted_image(
    key1_jerry_image, key2_jerry_image, key2_encrypted_jerry_image)
two_keys_decrypted_panda_image          = decrypt_two_keys_encrypted_image(
    key1_panda_image, key2_panda_image, key2_encrypted_panda_image)

two_key_decrypted_images_list         = [two_keys_decrypted_doraemon_image, tw
o_keys_decrypted_tom_and_jerry_image,
                                         two_keys_decrypted_inside_out_image,
two_keys_decrypted_blue_rose_image,
                                         two_keys_decrypted_boy_image, two_key
s_decrypted_flowers_image,
                                         two_keys_decrypted_idea_image, two_ke
ys_decrypted_jerry_image,
                                         two_keys_decrypted_panda_image]
two_key_decrypted_images_titles_list = ["Two Keys Decrypted Doraemon Image",
"Two Keys Decrypted Tom and Jerry Image",
                                         "Two Keys Decrypted Inside Out Image"
, "Two Keys Decrypted Blue Rose Image",
                                         "Two Keys Decrypted Boy Image", "Two
Keys Decrypted Flowers Image",
                                         "Two Keys Decrypted Idea Image", "Two
Keys Decrypted Jerry Image",
                                         "Two Keys Decrypted Panda Image"]

display_images_horizontaly(20, 20, 3, 3, two_key_decrypted_images_list, two_k
ey_decrypted_images_titles_list)
```

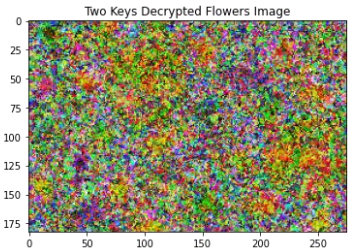


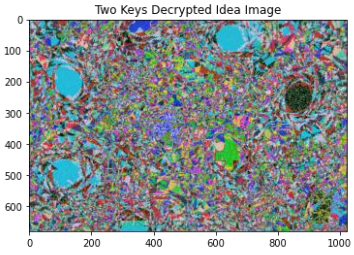


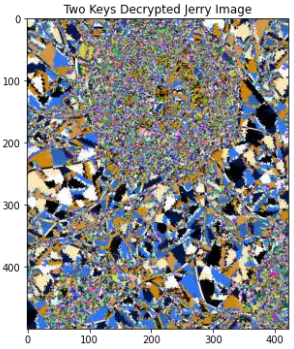


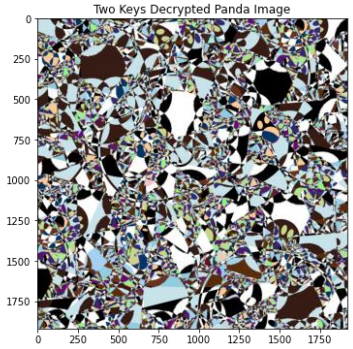






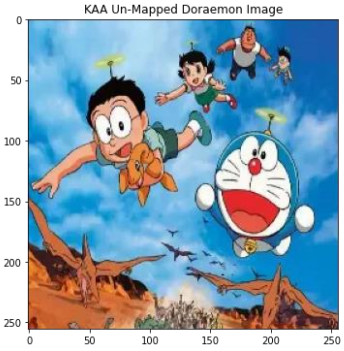


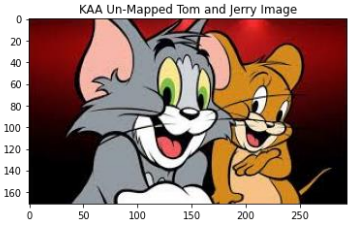


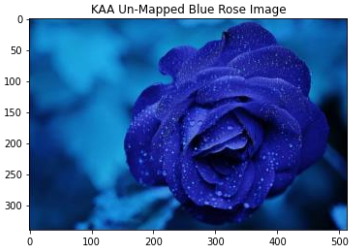


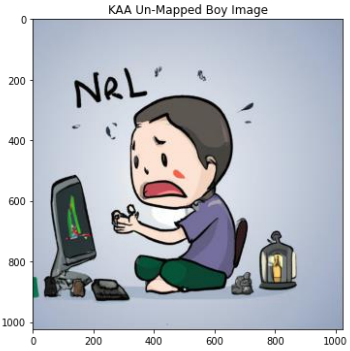

```
# Kaa Un-Mapping of images for getting back original pixels
kaa_unmapped_doraemon_image      = perform_combined_transformation_in_image(T
HETA, VELOCITY1, VELOCITY2, TIME, FREQUENCY,
                                                                        two
_keys_decrypted_doraemon_image, True)
kaa_unmapped_tom_and_jerry_image = perform_combined_transformation_in_image(T
HETA, VELOCITY1, VELOCITY2, TIME, FREQUENCY,
                                                                        two
_keys_decrypted_tom_and_jerry_image, True)
kaa_unmapped_inside_out_image    = perform_combined_transformation_in_image(T
HETA, VELOCITY1, VELOCITY2, TIME, FREQUENCY,
                                                                        two
_keys_decrypted_inside_out_image, True)
```

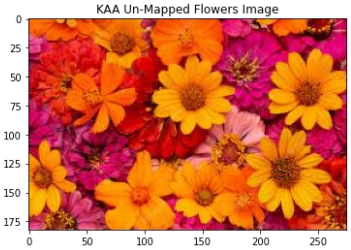
```
kaa_unmapped_blue_rose_image     = perform_combined_transformation_in_image(T
HETA, VELOCITY1, VELOCITY2, TIME, FREQUENCY,
                                                                        two
_keys_decrypted_blue_rose_image, True)
kaa_unmapped_boy_image           = perform_combined_transformation_in_image(T
HETA, VELOCITY1, VELOCITY2, TIME, FREQUENCY,
                                                                        two
_keys_decrypted_boy_image, True)
kaa_unmapped_flowers_image       = perform_combined_transformation_in_image(T
HETA, VELOCITY1, VELOCITY2, TIME, FREQUENCY,
                                                                        two
_keys_decrypted_flowers_image, True)
kaa_unmapped_idea_image          = perform_combined_transformation_in_image(T
HETA, VELOCITY1, VELOCITY2, TIME, FREQUENCY,
                                                                        two
_keys_decrypted_idea_image, True)
kaa_unmapped_jerry_image         = perform_combined_transformation_in_image(T
HETA, VELOCITY1, VELOCITY2, TIME, FREQUENCY,
                                                                        two
_keys_decrypted_jerry_image, True)
kaa_unmapped_panda_image         = perform_combined_transformation_in_image(T
HETA, VELOCITY1, VELOCITY2, TIME, FREQUENCY,
                                                                        two
_keys_decrypted_panda_image, True)

kaa_unmapped_images_list        = [kaa_unmapped_doraemon_image, kaa_unmapped_
tom_and_jerry_image,
                                   kaa_unmapped_inside_out_image, kaa_unmappe
d_blue_rose_image, kaa_unmapped_boy_image,
                                   kaa_unmapped_flowers_image, kaa_unmapped_i
dea_image, kaa_unmapped_jerry_image,
                                   kaa_unmapped_panda_image]
kaa_unmapped_images_titles_list = ["KAA Un-Mapped Doraemon Image", "KAA Un-Ma
pped Tom and Jerry Image",
                                   "KAA Un-Mapped Inside Out Image","KAA Un-M
apped Blue Rose Image",
                                   "KAA Un-Mapped Boy Image", "KAA Un-Mapped
Flowers Image","KAA Un-Mapped Idea Image",
                                   "KAA Un-Mapped Jerry Image", "KAA Un-Mappe
d Panda Image"]

display_images_horizontaly(20, 20, 3, 3, kaa_unmapped_images_list, kaa_unmapp
ed_images_titles_list)
```

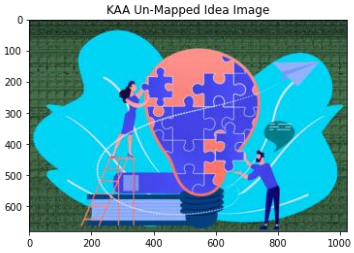


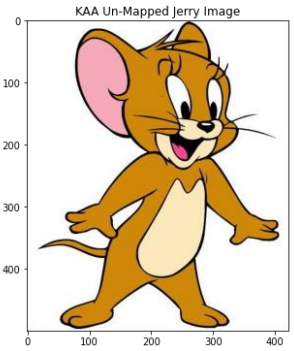


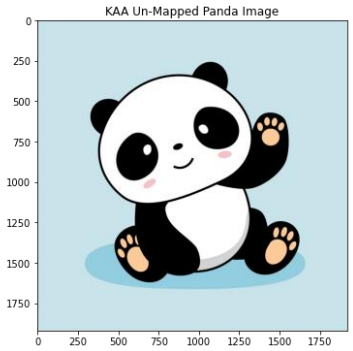










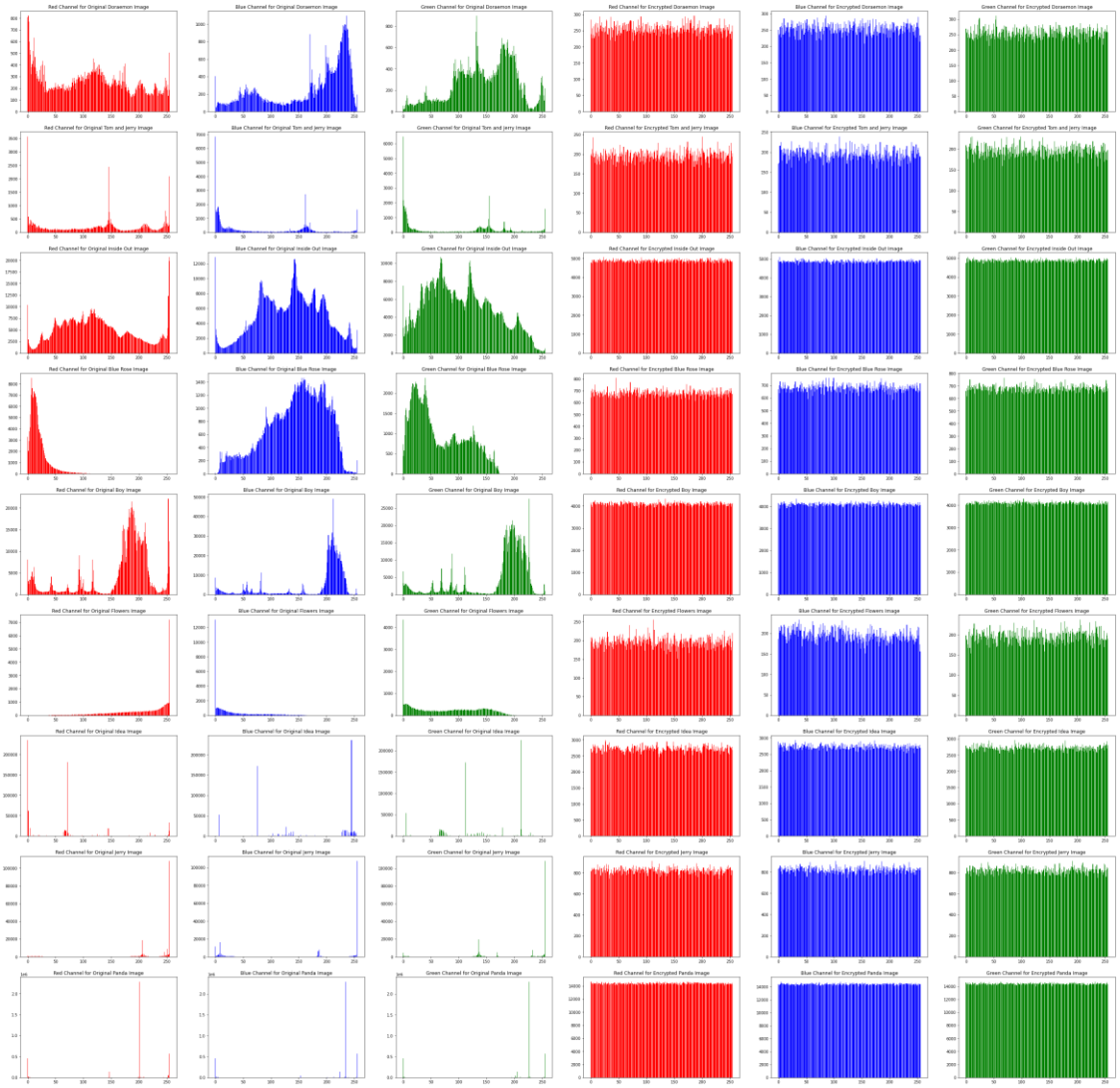



```
### VISUAL AND HISTOGRAM ANALYSES

# Function to convert a 2D array to 1D array for the purpose of vieweing a hi
stogram
# two_dimensional_array - The 2D array the needs to be converted into a 1D ar
ray
def convert_2D_array_to_1D_array(two_dimensional_array):
    x, y = len(two_dimensional_array), len(two_dimensional_array[0])
    return [two_dimensional_array[i][j] for i in range(x) for j in range(y)]

# Function to get frequency of different pixels in an array
# one_dimensional_array - 1D array containing all the pixels of a particular
channel in the range of [0,255]
def get_frequency_of_pixels(one_dimensional_array):
```

```
    pixels = [0 for each in range(256)]
    for each in one_dimensional_array:
        pixels[each] += 1
    return pixels

# Function to display histograms of tuples containing red, blue and green cha
nnels of an image
# This function helps in comparing between vieweing the color distribution of
original and encrypted image
# length                  - Length of the 2D histogram to be displayed
# breadth                 - Breadth of the 2D histogram to be displayed
# channles_original       - Contains list of tuples of the three RGB channles
of the original image
# in the order of red, blue and green
# channels_encrypted      - Contains list of tuples of the three RGB channels
of the encrypted image
# in the order of red, blue and green
# image_titles            - Contains the list of titles of image for which the
histogram is being displayed
def display_histograms_of_three_channels(length, breadth, channels_original,
channels_encrypted, image_titles):
    limit = len(image_titles)
    rcParams['figure.figsize'] = length, breadth
    total_pixels = [each for each in range(256)]
    figure, axis = plt.subplots(limit, 6)
    for x in range(limit):
        axis[x][0].bar(total_pixels, get_frequency_of_pixels(convert_2D_array
_to_1D_array(channels_original[x][0])),
                       color='red', linewidth=0.1)
        axis[x][0].set_title("Red Channel for Original {}".format(image_title
s[x]))
        axis[x][1].bar(total_pixels, get_frequency_of_pixels(convert_2D_array
_to_1D_array(channels_original[x][1])),
                       color='blue', linewidth=0.1)
        axis[x][1].set_title("Blue Channel for Original {}".format(image_titl
es[x]))
        axis[x][2].bar(total_pixels, get_frequency_of_pixels(convert_2D_array
_to_1D_array(channels_original[x][2])),
                       color='green', linewidth=0.1)
        axis[x][2].set_title("Green Channel for Original {}".format(image_tit
les[x]))
        axis[x][3].bar(total_pixels, get_frequency_of_pixels(convert_2D_array
_to_1D_array(channels_encrypted[x][0])),
                         color='red', linewidth=0.1)
        axis[x][3].set_title("Red Channel for Encrypted {}".format(image_titl
es[x]))
```

```
        axis[x][4].bar(total_pixels, get_frequency_of_pixels(convert_2D_array
_to_1D_array(channels_encrypted[x][1])),
                       color='blue', linewidth=0.1)
        axis[x][4].set_title("Blue Channel for Encrypted {}".format(image_tit
les[x]))
        axis[x][5].bar(total_pixels, get_frequency_of_pixels(convert_2D_array
_to_1D_array(channels_encrypted[x][2])),
                       color='green', linewidth=0.1)
        axis[x][5].set_title("Green Channel for Encrypted {}".format(image_ti
tles[x]))
    return None

# Original and Encrypted Channels for different Images
channels_encrypted_doraemon_image, channels_original_doraemon_image
= \
(red_encrypted_doraemon_image, blue_encrypted_doraemon_image, green_encrypted
_doraemon_image), \
(red_doraemon_image, blue_doraemon_image, green_doraemon_image)
channels_encrypted_tom_and_jerry_image, channels_original_tom_and_jerry_image
= \
(red_encrypted_tom_and_jerry_image, blue_encrypted_tom_and_jerry_image, green
_encrypted_tom_and_jerry_image), \
(red_tom_and_jerry_image, blue_tom_and_jerry_image, green_tom_and_jerry_image
)
channels_encrypted_inside_out_image, channels_original_inside_out_image
= \
(red_encrypted_inside_out_image, blue_encrypted_inside_out_image, green_encry
pted_inside_out_image), \
(red_inside_out_image, blue_inside_out_image, green_inside_out_image)
channels_encrypted_blue_rose_image, channels_original_blue_rose_image
= \
(red_encrypted_blue_rose_image, blue_encrypted_blue_rose_image, green_encrypt
ed_blue_rose_image), \
(red_blue_rose_image, blue_blue_rose_image, green_blue_rose_image)
channels_encrypted_boy_image, channels_original_boy_image
= \
(red_encrypted_boy_image, blue_encrypted_boy_image, green_encrypted_boy_image
), \
(red_boy_image, blue_boy_image, green_boy_image)
channels_encrypted_flowers_image, channels_original_flowers_image
= \
(red_encrypted_flowers_image, blue_encrypted_flowers_image, green_encrypted_f
lowers_image), \
(red_flowers_image, blue_flowers_image, green_flowers_image)
channels_encrypted_idea_image, channels_original_idea_image
= \
```

```
(red_encrypted_idea_image, blue_encrypted_idea_image, green_encrypted_idea_im
age), \
(red_idea_image, blue_idea_image, green_idea_image)
channels_encrypted_jerry_image, channels_original_jerry_image
= \
(red_encrypted_jerry_image, blue_encrypted_jerry_image, green_encrypted_jerry
_image), \
(red_jerry_image, blue_jerry_image, green_jerry_image)
channels_encrypted_panda_image, channels_original_panda_image
= \
(red_encrypted_panda_image, blue_encrypted_panda_image, green_encrypted_panda
_image), \
(red_panda_image, blue_panda_image, green_panda_image)

channels_encrypted = [channels_encrypted_doraemon_image, channels_encrypted_t
om_and_jerry_image,
                      channels_encrypted_inside_out_image, channels_encrypted
_blue_rose_image, channels_encrypted_boy_image,
                      channels_encrypted_flowers_image, channels_encrypted_id
ea_image, channels_encrypted_jerry_image,
                      channels_encrypted_panda_image]
channels_original  = [channels_original_doraemon_image, channels_original_tom
_and_jerry_image,
                      channels_original_inside_out_image, channels_original_b
lue_rose_image, channels_original_boy_image,
                      channels_original_flowers_image, channels_original_idea
_image, channels_original_jerry_image,
                      channels_original_panda_image]
image_titles       = ["Doraemon Image", "Tom and Jerry Image", "Inside Out Im
age", "Blue Rose Image", "Boy Image",
                      "Flowers Image", "Idea Image", "Jerry Image", "Panda Im
age"]

display_histograms_of_three_channels(50, 50, channels_original, channels_encr
ypted, image_titles)
```



```
### INFORMATION ENTROPY

# Function to get information entropy of encrypted channels
# channels_encrypted - List of tuples of rgb channels of encrypted images
# image_titles       - List of titles for the images
def get_information_entropy(channels_encrypted, image_titles):
    def get_entropy(channel):
        _bins = 256
        hist, _ = np.histogram(channel.ravel(), bins=_bins, range=(0, _bins))
        prob_dist = hist / hist.sum()
        return entropy(prob_dist, base=2)
    x = len(image_titles)
    e = {"Red Channel": [], "Blue Channel": [], "Green Channel": [], "Image":
[]}
```

```
    for i in range(x):
        e["Red Channel"].append(get_entropy(channels_encrypted[i][0]))
        e["Blue Channel"].append(get_entropy(channels_encrypted[i][1]))
        e["Green Channel"].append(get_entropy(channels_encrypted[i][2]))
        e["Image"].append(get_entropy(cv2.merge(
            [channels_encrypted[i][0], channels_encrypted[i][2], channels_enc
rypted[i][1]])))
    df = pd.DataFrame(e)
    df.index = image_titles
    return df

information_entropy_df_of_encrypted_image = get_information_entropy(channels_
encrypted, image_titles)
information_entropy_df_of_original_image  = get_information_entropy(channels_
original,  image_titles)
print("################################################################################
#########")
print("Encrypted Image")
print("################################################################################
#########")
print(information_entropy_df_of_encrypted_image)
print("################################################################################
#########")
print("Original Image")
print("################################################################################
#########")
print(information_entropy_df_of_original_image)
print("################################################################################
#########")
```

```
################################################################################
##
Encrypted Image
################################################################################
##
                     Red Channel  Blue Channel  Green Channel     Image
Doraemon Image          7.997365      7.997156       7.996973  7.999091
Tom and Jerry Image     7.996433      7.996289       7.996704  7.998772
Inside Out Image        7.999843      7.999864       7.999865  7.999951
Blue Rose Image         7.998881      7.998763       7.998973  7.999574
Boy Image               7.999782      7.999794       7.999843  7.999937
Flowers Image           7.996473      7.996012       7.995753  7.998907
Idea Image              7.999156      7.999476       7.999436  7.999753
Jerry Image             7.999165      7.999153       7.999032  7.999686
Panda Image             7.999950      7.999950       7.999948  7.999976
################################################################################
```

```
##
Original Image
################################################################################
##
                     Red Channel  Blue Channel  Green Channel     Image
Doraemon Image          7.899040      7.542754       7.629843  7.927936
Tom and Jerry Image     7.381227      6.404629       6.462237  6.982173
Inside Out Image        7.802691      7.727499       7.793973  7.860580
Blue Rose Image         5.569047      7.621424       7.223039  7.468365
Boy Image               6.970757      6.446514       6.803799  7.054215
Flowers Image           6.709565      5.768534       7.264404  7.336660
Idea Image              3.341149      3.522701       3.652871  4.960531
Jerry Image             3.910089      3.789968       3.813268  4.212389
Panda Image             1.990224      1.988982       1.970239  3.054637
################################################################################
##

original_images  = [rgb_doraemon_image, rgb_tom_and_jerry_image, rgb_inside_o
ut_image, rgb_blue_rose_image, rgb_boy_image,
                    rgb_flowers_image, rgb_idea_image, rgb_jerry_image, rgb_p
anda_image]
encrypted_images = [key2_encrypted_doraemon_image, key2_encrypted_tom_and_jer
ry_image, key2_encrypted_inside_out_image,
                    key2_encrypted_blue_rose_image, key2_encrypted_boy_image,
key2_encrypted_flowers_image,
                    key2_encrypted_idea_image, key2_encrypted_jerry_image, ke
y2_encrypted_panda_image]
image_titles       = ["Doraemon Image", "Tom and Jerry Image", "Inside Out Im
age", "Blue Rose Image", "Boy Image",
                      "Flowers Image", "Idea Image", "Jerry Image", "Panda Im
age"]

### MEAN SQUARED ERROR

# Function to calculate mean squared error between original and encrypted ima
ges
# original_images  - List of orignal images
# encrypted_images - List of encrypted images
# image_titles     - List of titles for the images
def get_mean_sqaured_error(original_images, encrypted_images, image_titles):
    def error(original_image, encrypted_image):
        error = np.sum((original_image.astype("float") - encrypted_image.asty
pe("float")) ** 2)
        error /= float(original_image.shape[0] * encrypted_image.shape[1])
        return error
    df = pd.DataFrame()
    df['MSE'] = [error(original_images[i], encrypted_images[i]) for i in rang
```

```
e(len(image_titles))]
    df.index = image_titles
    return df

mean_squared_error_df = get_mean_sqaured_error(original_images, encrypted_ima
ges, image_titles)
print("##########################################################################
#########")
print("Mean Squared Error")
print("##########################################################################
#########")
print(mean_squared_error_df)
print("##########################################################################
#########")

##########################################################################
##
Mean Squared Error
##########################################################################
##
                            MSE
Doraemon Image       32294.451660
Tom and Jerry Image  42382.251542
Inside Out Image     27786.351954
Blue Rose Image      37549.497007
Boy Image            34731.964033
Flowers Image        42296.647968
Idea Image           40522.690314
Jerry Image          54125.648346
Panda Image          48954.977058
##########################################################################
##

### PEAK SIGNAL TO NOISE RATIO

# Function to calculate peak signal to noise ratio
# original_images  - List of orignal images
# encrypted_images - List of encrypted images
# image_titles     - List of titles for the images
def get_peak_signal_to_noise_ratio(original_images, encrypted_images, image_t
itles):
    df = pd.DataFrame()
    df['PSNR'] = [cv2.PSNR(original_images[i], encrypted_images[i]) for i in
range(len(image_titles))]
    df.index = image_titles
    return df
```

```
peak_signal_to_noise_ratio_df = get_peak_signal_to_noise_ratio(original_image
s, encrypted_images, image_titles)
print("##############################################################################
#########")
print("Peak Signal To Noise Ratio")
print("##############################################################################
#########")
print(peak_signal_to_noise_ratio_df)
print("##############################################################################
#########")

##############################################################################
##
Peak Signal To Noise Ratio
##############################################################################
##
                         PSNR
Doraemon Image       7.810737
Tom and Jerry Image  6.630176
Inside Out Image     8.463701
Blue Rose Image      7.155975
Boy Image            7.494723
Flowers Image        6.638957
Idea Image           6.825033
Jerry Image          5.567985
Panda Image          6.004048
##############################################################################
##
```

```
### CORRELATION COEFFICIENT ANALYSIS

# Function to get correlation coefficient between original images and encrypt
ed images
# original_images  - List of original images
# encrypted_images - List of encrypted images
# image_titles     - List of titles for the images
def get_correlation_coefficient_analysis(original_images, enrypted_images, im
age_titles):
    x = len(image_titles)
    df = pd.DataFrame()
    df['correlation coefficient'] = [np.corrcoef(original_images[i].flat, enc
rypted_images[i].flat)[1][0] for i in range(x)]
    df.index = image_titles
    return df

correlation_coefficient_analysis_df = get_correlation_coefficient_analysis(or
iginal_images, encrypted_images, image_titles)
```

```
print("##############################################################################
#########")
print("Correlation Coefficient Analysis")
print("##############################################################################
#########")
print(correlation_coefficient_analysis_df)
print("##############################################################################
#########")
```

```
##############################################################################
##
Correlation Coefficient Analysis
##############################################################################
##
                     correlation coefficient
Doraemon Image                      0.001120
Tom and Jerry Image                -0.006055
Inside Out Image                    0.000047
Blue Rose Image                    -0.002505
Boy Image                          -0.001050
Flowers Image                      -0.009307
Idea Image                         -0.000402
Jerry Image                        -0.002517
Panda Image                        -0.000469
##############################################################################
##
```

CONCLUSION

As is evident from the various security analysis, the proposed algorithm suits to withstand various cryptanalytic attacks, including visual, statistical, differential and brute-force attacks.

This can hence be used as a tool for encrypting various multimedia images and thus securing the information. The future scope of work would include adding an extra layer of chaos to the generation of key by using a one-dimensional chaotic function to create chaos in the control parameters as used in the different chaotic functions for the purpose of encryption.

REFERENCES

[1] A. Al-Khedhairi, A. Elsonbaty, A. A. Elsadany, and E. A. A. Hagras,"Hybrid cryptosystem based on pseudo chaos of novel fractional order map and elliptic curves," IEEE Access, vol. 8, pp. 57733–57748, 2020.

[2] W. Alexan, M. ElBeltagy, and A. Aboshousha, "Image encryption through Lucas sequence, S-Box and chaos theory," in Proc. 8th NAFOSTED Conf. Inf. Comput. Sci. (NICS), Dec. 2021, pp. 77–83.

[3] W. Alexan, M. ElBeltagy, and A. Aboshousha, "RGB image encryption through cellular automata, S-Box and the Lorenz system," Symmetry, vol. 14, no. 3, p. 443, Feb. 2022.

[4] T. S. Ali and R. Ali, "A new chaos based color image encryption algorithm using permutation substitution and Boolean operation," Multimedia Tools Appl., vol. 79, nos. 27–28, pp. 19853–19873, Jul. 2020.

[5] D. R. Anderson, Model Based Inference in the Life Sciences: A Primer on Evidence, vol. 31. New York, NY, USA: Springer, 2008.

[6] B. Arpacı, E. Kurt, K. Çelik, and B. Ciylan, "Colored image encryption and decryption with a new algorithm and a hyperchaotic electrical circuit," J. Electr. Eng. Technol., vol. 15, no. 3, pp. 1413–1429, May 2020.

[7] D. Arroyo, J. Diaz, and F. B. Rodriguez, "Cryptanalysis of a one round chaos-based substitution permutation network," Signal Process., vol. 93, no. 5, pp. 1358–1364, May 2013.

[8] M. Bañados and I. Reyes, "A short review on Noether's theorems, gauge symmetries and boundary terms," Int. J. Modern Phys. D, vol. 25, no. 10, Sep. 2016, Art. no. 1630021.

[9] T. Bertschinger, N. Flowers, S. Moseley, C. Pfeifer, J. Tasson, and S. Yang,"Spacetime symmetries and classical mechanics," Symmetry, vol. 11, no. 1, p. 22, Dec. 2018.

[10] D. J. Driebe, Fully Chaotic Maps and Broken Time Symmetry, vol. 4. New York, NY, USA: Springer, 1999.

[11] H. Gao and X. Wang, "Chaotic image encryption algorithm based on zigzag transform with bidirectional crossover from random position," IEEE Access, vol. 9, pp. 105627–105640, 2021.

[12] H. Garcés and B. C. Flores, "Statistical analysis of Bernoulli, logistic, and tent maps with applications to radar signal design," Proc. SPIE, vol. 6210, May 2006, Art. no. 62100G.

[13] B. Ge, X. Chen, G. Chen, and Z. Shen, "Secure and fast image encryption algorithm using Hyper-Chaos-Based key generator and vector operation," IEEE Access, vol. 9, pp. 137635–137654, 2021.

[14] L. Gong, K. Qiu, C. Deng, and N. Zhou, "An image compression and encryption algorithm based on chaotic system and compressive sensing," Opt. Laser Technol., vol. 115, pp. 257–267, Jul. 2019.

[15] E. Hasanzadeh and M. Yaghoobi, "A novel color image encryption algorithm based on substitution box and hyper-chaotic system with fractal keys," Multimedia Tools Appl., vol. 79, pp. 1–19, Mar. 2019.

[16] X. Hu, L. Wei, W. Chen, Q. Chen, and Y. Guo, "Color image encryption algorithm based on dynamic chaos and matrix convolution," IEEE Access, vol. 8, pp. 12452–12466, 2020.

[17] Z. Hua, Y. Zhou, C.-M. Pun, and C. L. P. Chen, "Image encryption using 2D logistic-sine chaotic map," in Proc. IEEE Int. Conf. Syst., Man, Cybern. (SMC), Oct. 2014, pp. 3229–3234.

[18] L. M. Jawad, "A new scan pattern method for color image encryption based on 3D-Lorenzo chaotic map method," Multimedia Tools Appl., vol. 80, no. 24, pp. 33297–33312, Oct. 2021. 11552 VOLUME 11, 2023 W. Alexan et al.: Color Image Encryption Through Chaos and KAA Map [19] K. C. Jithin and S. Sankar, "Colour image encryption algorithm combining Arnold map, DNA sequence operation, and a Mandelbrot set," J. Inf. Secur. Appl., vol. 50, Feb. 2020, Art. no. 102428.

[20] M. Khan and F. Masood, "A novel chaotic image encryption technique based on multiple discrete dynamical maps," Multimedia Tools Appl., vol. 78, no. 18, pp. 26203–26222, Sep. 2019.

[21] M. Khan and T. Shah, "An efficient chaotic image encryption scheme," Neural Comput. Appl., vol. 26, no. 5, pp. 1137–1148, Jul. 2015.

[22] D. E. Knuth, The Art of Computer Programming. Volume 1: Fundamental Algorithms. Volume 2: Seminumerical Algorithms. Providence, Rhode Island: Bulletin of the American Mathematical Society, 1997.

[23] V. Kumar and A. Girdhar, "A 2D logistic map and Lorenz-Rossler chaotic system based RGB image encryption approach," Multimedia Tools Appl., vol. 80, no. 3, pp. 3749–3773, 2021.

[24] M. Kumari and S. Gupta, "Performance comparison between chaos and quantum-chaos based image encryption techniques," Multimedia Tools Appl., vol. 80, no. 24, pp. 33213–33255, Oct. 2021.

[25] B. Li, X. Liao, and Y. Jiang, "A novel image encryption scheme based on logistic map and dynatomic modular curve," Multimedia Tools Appl., vol. 77, no. 7, pp. 8911–8938, Apr. 2018.

[26] H. Liu and A. Kadir, "Asymmetric color image encryption scheme using 2D discrete-time map," Signal Process., vol. 113, pp. 104–112, Aug. 2015.

[27] H. Liu, X. Wang, and A. Kadir, "Chaos-based color image encryption using one-time keys and choquet fuzzy integral," Int. J. Nonlinear Sci. Numer. Simul., vol. 15, no. 1, pp. 1–10, Feb. 2014.

[28] K. S. Mallesh, S. Chaturvedi, V. Balakrishnan, R. Simon, and N. Mukunda, "Symmetries and conservation laws in classical and quantum mechanics," Resonance, vol. 16, no. 3, pp. 254–273, Mar. 2011.

[29] A. Y. Niyat, M. H. Moattar, and M. N. Torshiz, "Color image encryption based on hybrid hyper-chaotic system and cellular automata," Opt. Lasers Eng., vol. 90, pp. 225–237, Mar. 2017.

[30] M. E. O'Neill, "PCG: A family of simple fast space-efficient statistically good algorithms for random number generation," Harvey Mudd College, Claremont, CA, USA, Tech. Rep. HMC-CS-2014-0905, Sep. 2014.

[Online]. Available: https://www.cs.hmc.edu/tr/hmc-cs-2014-0905.pdf

[31] A. Rotenberg, "A new pseudo-random number generator," J. ACM, vol. 7, no. 1, pp. 75–77, Jan. 1960.

[32] C. E. Shannon, "Communication theory of secrecy systems," Bell Syst. Tech. J., vol. 28, pp. 656–715, Oct. 1949.

[33] C. E. Shannon, "A mathematical theory of communication," Bell Syst. Tech. J., vol. 27, no. 3, pp. 379–423, Jul./Oct. 1948.

[34] N. Ben Slimane, N. Aouf, K. Bouallegue, and M. Machhout, "A novel chaotic image cryptosystem based on DNA sequence operations and single neuron model," Multimedia Tools Appl., vol. 77, no. 23, pp. 30993–31019, Dec. 2018.

[35] W. Stallings, Cryptography & Network Security GE. London, U.K.: Pearson, 2017.

[36] H.-C. Tang, "An analysis of linear congruential random number generators when multiplier restrictions exist," Eur. J. Oper. Res., vol. 182, no. 2, pp. 820–828, Oct. 2007.

[37] M. Tanveer, T. Shah, A. Rehman, A. Ali, G. F. Siddiqui, T. Saba, and U. Tariq, "Multi-images encryption scheme based on 3D chaotic map and substitution box," IEEE Access, vol. 9, pp. 73924–73937, 2021.

[38] A. U. Rehman, X. Liao, R. Ashraf, S. Ullah, and H. Wang, "A color image encryption technique using exclusive-OR with DNA complementary rules based on chaos theory and SHA-2," Optik, vol. 159, pp. 348–367, Apr. 2018.

[39] M. V. Harten, "The dynamics of the one-dimensional tent map family and quadratic family," M.S. thesis, Dept. Math. Appl. Math., Faculty Sci. Eng., Univ. Groningen, The Netherlands, 2018.

[40] X.-Y. Wang and Z.-M. Li, "A color image encryption algorithm based on Hopfield chaotic neural network," Opt. Lasers Eng., vol. 115, pp. 107–118, Apr. 2019.

[41] Y. Wang, C. Wu, S. Kang, Q. Wang, and V. Mikulovich, "Multi-channel chaotic encryption algorithm for color image based on dna coding," Multimedia Tools Appl., vol. 79, pp. 1–26, Jul. 2020.

[42] X. Wu, K. Wang, X. Wang, and H. Kan, "Lossless chaotic color image cryptosystem based on DNA encryption and entropy," Nonlinear Dyn., vol. 90, no. 2, pp. 855–875, Oct. 2017.

[43] X. Wu, K. Wang, X. Wang, H. Kan, and J. Kurths, "Color image DNA encryption using NCA map-based CML and one-time keys," Signal Process., vol. 148, pp. 272–287, Jul. 2018.

[44] L. Xu, Z. Li, J. Li, and W. Hua, "A novel bit-level image encryption algorithm based on chaotic maps," Opt. Lasers Eng., vol. 78, pp. 17–25, Mar. 2016.

[45] B. Yang and X. Liao, "A new color image encryption scheme based on logistic map over the finite field ZN," Multimedia Tools Appl., vol. 77, no. 16, pp. 21803–21821, Aug. 2018.

[46] F. Yang, J. Mou, K. Sun, Y. Cao, and J. Jin, "Color image compressionencryption algorithm based on fractional-order memristor chaotic circuit," IEEE Access, vol. 7, pp. 58751–58763, 2019.

[47] I. Younas and M. Khan, "A new efficient digital image encryption based on inverse left almost semi group and Lorenz chaotic system," Entropy, vol. 20, no. 12, p. 913, 2018.

[48] X. Zhang and X. Wang, "Multiple-image encryption algorithm based on DNA encoding and chaotic system," Multimedia Tools Appl., vol. 78, no. 6, pp. 7841–7869, 2019.

[49] X. Zhang, L. Wang, Y. Wang, Y. Niu, and Y. Li, "An image encryption algorithm based on hyperchaotic system and variable-step Josephus problem," Int. J. Opt., vol. 2020, pp. 1–15, Oct. 2020.

[50] Y.-Q. Zhang, Y. He, P. Li, and X.-Y. Wang, "A new color image encryption scheme based on 2DNLCML system and genetic operations," Opt. Lasers Eng., vol. 128, May 2020, Art. no. 106040.

[51] C. Zou, Q. Zhang, X. Wei, and C. Liu, “Image encryption based on improved Lorenz system,” IEEE Access, vol. 8, pp. 75728–75740, 2020.
[52] WASSIM ALEXAN , (Senior Member, IEEE), MARWA ELKANDOZ, MAGGIE MASHALY , (Senior Member, IEEE), EMAN AZAB, (Senior Member, IEEE),AND AMR ABOSHOUSHA, “Color Image Encryption Through Chaos and KAA Map”