MA203

Numerical Analysis of Encryption/Decryption using Chaos Theory

Numerical Methods - Project

Online Available: Image

Group 58

1. Aditya Kumar (22110015)
2. Archit Dhakar (22110031)
3. Aryan Sahu (22110038)

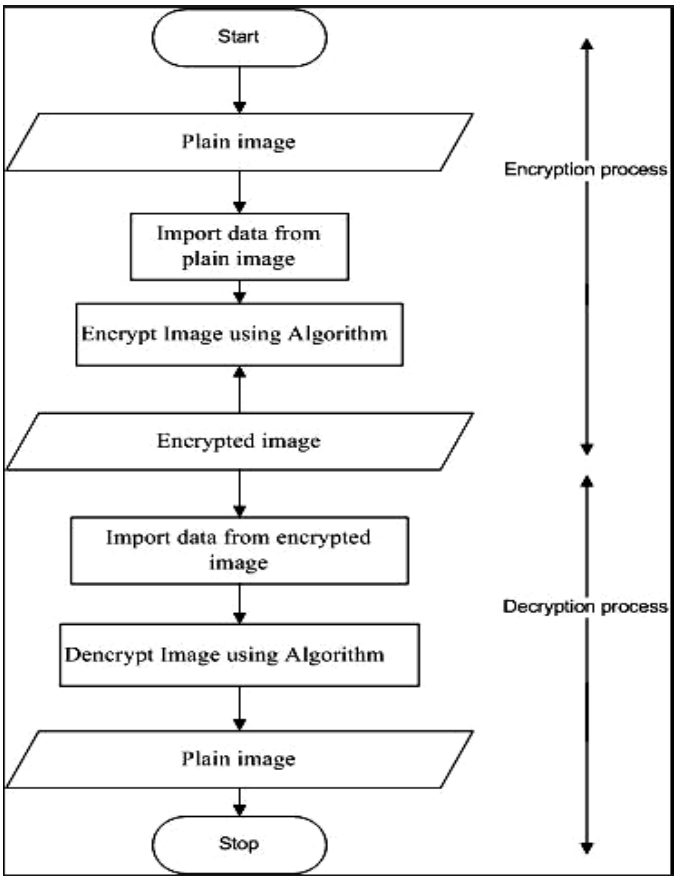
Problem Statement

Encryption can be loosely described as the conversion of a data type or information that needs to be relayed over a transmission channel to some format that will be unrecognisable to any outside person. Encryption is an important topic of study because it enables us to transmit data or information safely and privately without interference or eavesdropping. All communication systems nowadays rely on encryption techniques to secure their or their users' conversations. Other applications include ground and space transmission, military services and other governmental functions.

Extensive research is being conducted to explore various encryption techniques which can be used to protect data from getting leaked. Some desirable features include computational ease, time of encryption, level of security, difficulty for a breach, and safety. One attractive candidate is the use of chaos theory because of its extraordinary random behaviour. Chaos theory, however, is a new method which needs more development and computation power. With the advent of high-power computers, it is now possible to use the chaos theory as a mode of encryption and to establish better transmission media.

The Chaos theory involves a set of iterative equations that give random solutions over different iterations and other initial conditions. These equations often govern many real-life scenarios, such as the population of a species. The use of these equations in the method of encryption is done as follows: The data or information needed to be transmitted is converted to numerical format, i.e., it can be represented using a specific and unique set of numbers. These numbers are then set as the initial conditions or number of iterations for the groups of iterative equations. The result we get from this is totally unpredictable unless you know the conditions and the encryption technique. This result is then relayed as the encrypted message and then decrypted at the receiving end by reversing the above procedure.

The issue with using the Chaos theory is that the method requires very high computational power, and the values often take a lot of computation time. This can be reduced by analysing and establishing faster numerical methods for solving these equations, reducing the computation time and allowing the use of this method in real-world problems.



This project aims to successfully encrypt and decrypt an image signal using the method of Chaos Theory. A theoretical and mathematical model of the encryption technique will be developed, and the resulting system of equations will be solved numerically on a computer. The specific objectives are presented below:

1. Develop a model of encryption technique, which will be based on the Chaos theory equations.
2. Make suitable assumptions and make the problem tractable.
3. Derive the governing equations from the first principles of Chaos Theory.
4. Formulate the initial conditions required for the proposed equations to show random behaviour.
5. Choose the correct set of property values and parameters.
6. Adopt efficient numerical methods to reduce the time of computation.
7. Write a computer program to perform the encryption method for a sample image signal.
8. Successfully retrieve the original signal by decrypting the encryption using the reverse procedure.
9. Determine the computational time and efficiency of the method

1 Encryption/Decryption Model

1.1 Existing Models

Encryption and Decryption processes involve converting the given set of data or information into some unrecognisable format which would provide assurance that the data cannot be leaked while transmitting. This process has been done in various ways earlier. One example of such a method is the Block Cipher. The block cipher splits data into chunks and then encrypts them individually by using security keys.

The above-mentioned conventional methods have some drawbacks. These ciphers lack security efficiency. Many hackers have devised various ways to decrypt these encryptions because of their pre-defined behaviour. Also, these methods give the same ciphers whenever the same data is encrypted. This allows the hackers to devise a look-up table with ciphers for various kinds of data, which could help in forced decryption of data.

1.2 Proposed Model

The above mentioned issues are due to the lack of randomization in the implementation of the encryption technique. Therefore, to devise better encryption methods, we need to find more random behaviour.

If we observe around us, we will find various random happenings. Many systems behave so randomly that they cannot be predicted in any manner. The behaviour of these systems is termed as Chaotic. A Chaotic system's behaviour is governed solely by their nature and the initial conditions. Any small change in the initial conditions gets drastically magnified to change the result completely. This random behaviour can therefore be used as Encryption Technique for a safer data transmission.

The key to our method is the use of Chaotic maps as a method to encode the data. The chaotic maps involve a set of equations that govern the chaotic behaviour based on the initial conditions. These equations are obtained and implemented in the encryption technique using Numerical methods. The sequence we receive on iterating over these equations numerically is called a chaotic sequence.

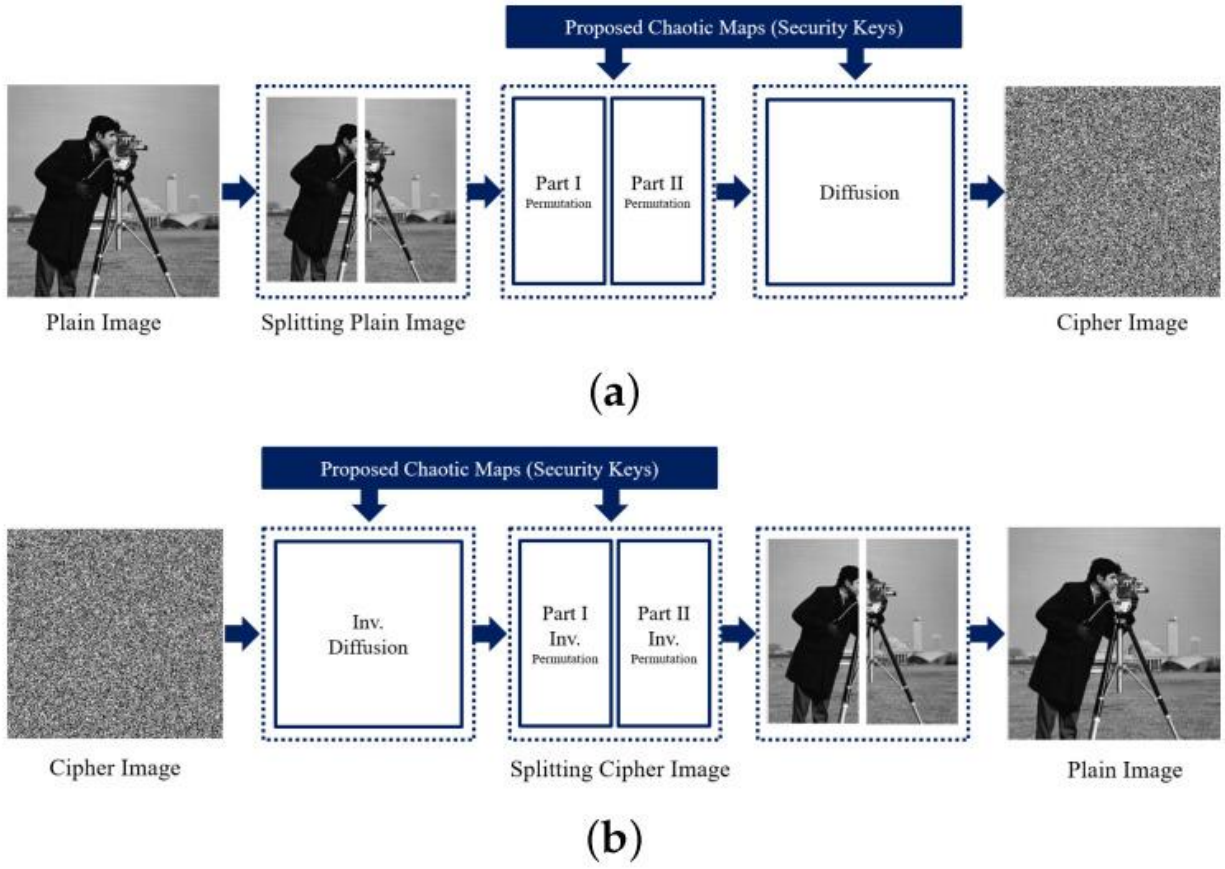


Figure 1: Image Encryption algorithm using chaotic maps.

Here, we will be devising an encryption technique for an image signal which can further be extrapolated for other signals as well. Our Encryption Model involves the following steps (as visible in figure 1) :

a. Image import and Formatting: This step involves converting the image signal into processable format.
b. Image splitting: We split the image in two parts for a more complex encryption.
c. Chaotic Map Scrambling: We scramble the data in the two parts according to a chaotic map and sequences. These sequences are generated using specific initial conditions. Then the given data is permuted based on some relation with the chaotic sequences which depends on the type of algorithm. This randomises the data sequence but in a specific manner known only for the encryption method.

d. Recombination of the two halves: After scrambling of the two halves of the data, the two are combined to again form a complete signal.
e. Second Layer Encryption: This involves adding a layer of normal encryption to ensure complete safety. In this step, we generate a random security key and use it as an encryption medium for the previously obtained signal.

This second layer of the encryption completes our encrypted signal and we have the cipher image ready to be transmitted.

On the recipient side, the process of decryption will take place. Decryption will require the same security key as generated earlier and the set of chaotic systems and the initial conditions to recover the original image. Decryption involves going through the same steps in the reverse order undoing each encryption step to obtain the original image.

2 Derivation of Model Equations

The proposed encryption and decryption model depends on the choice of chaotic system. In order to develop a chaotic system, we have used the Lorenz system of equations. The Lorenz system is a system of ordinary differential equations. The solutions of these ODE's are the basis for our chaotic map equations.

The Lorenz system ODE's are,

$$\frac{dx}{dt} = \sigma(y - x) \tag{1}$$

$$\frac{dy}{dt} = x(\rho - z) - y \tag{2}$$

$$\frac{dz}{dt} = xy - \beta z \tag{3}$$

In the real world these equations relate the properties of a two-dimensional fluid layer uniformly warmed from below and cooled from above.

Here $\sigma = 10$, $\rho = 28$, $\beta = 8/3$ are the constant system parameters.

3 Numerical Approach

The three ODEs of Lorenz can now be solved using a numerical method. In this case we are going to use Runge Kutta's fourth order method to solve the set of equations. We are using this method because of its high accuracy as compared to other numerical methods.

Runge Kutta's method is an iterative method which includes the Euler method, for finding approximate solutions of nonlinear equations in each iteration.

In this method we have to give the initial value conditions for the variables involved in the equations. We also have to provide the step size h to use this method.

In fourth order Runge Kutta's method we have,

The general form of an ODE equation is,

$$\frac{dy}{dx} = f(x_i , y_i)$$

where we can write,

$$y_{i+1} = y_i + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)h$$

$$k_1 = f(x_i , y_i)$$

$$k_2 = f(x_i+\frac{h}{2} , y_i+ k1\frac{h}{2})$$

$$k_3 = f(x_i+\frac{h}{2} , y_i+ k2\frac{h}{2})$$

$$k_4 = f(x_i+h , y_i+k_3h)$$

Thus, for our Lorenz system of ODEs we have,

$$\frac{dx}{dt} = f(x_i , y_i)= \sigma(y - x)$$

$$\frac{dy}{dt} = g(x_i , y_i , z_i) = x(\rho - z) - y$$

$$\frac{dz}{dt} = h(x_i , y_i , z_i) = xy - \beta z$$

Thus,

$k_1 = [\, f(x_i, y_i),\ g(x_i, y_i, z_i),\ h(x_i, y_i, z_i)\,]$

$k_2 = [\, f(x_i+\frac{h}{2}, y_i+k_1\frac{h}{2}),\ g(x_i+\frac{h}{2}, y_i+k_1\frac{h}{2}, z_i+k_1\frac{h}{2}),\ h(x_i+\frac{h}{2}, y_i+k_1\frac{h}{2}, z_i+k_1\frac{h}{2})\,]$

$k_3 = [\, f(x_i+\frac{h}{2}, y_i+k_2\frac{h}{2}),\ g(x_i+\frac{h}{2}, y_i+k_2\frac{h}{2}, z_i+k_2\frac{h}{2}),\ h(x_i+\frac{h}{2}, y_i+k_2\frac{h}{2}, z_i+k_2\frac{h}{2})\,]$

$k_4 = [f(x_i+h, y_i+k_3h),\ g(x_i+h, y_i+k_3h, z_i+k_3h),\ h(x_i+h, y_i+k_3h, z_i+k_3h)\,]$

$\sigma = 10, \rho = 28, \beta = 8/3$. Now using the iteration algorithm with the initial condition we can find the next value of the variable. The values we have got from the iterative approach for variables x,y,z can be used to create chaotic sequences.

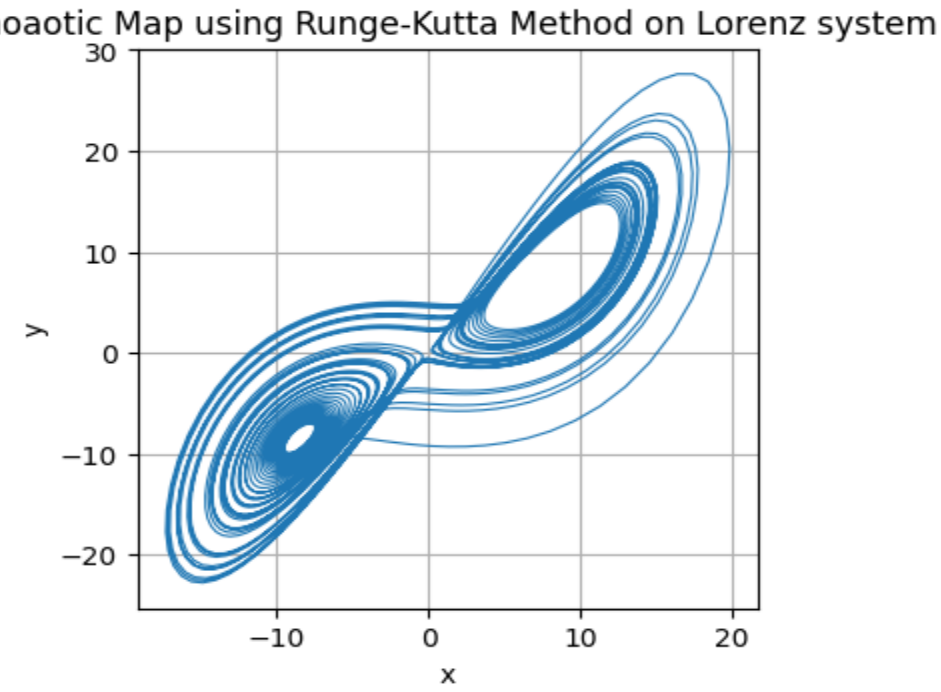


Fig 2. Chaotic plot for x and y

The above plot shows the chaotic behaviour using x,y to create chaotic sequences. Thus, we can say that solution of lorenz equations with $\sigma = 10, \rho = 28, \beta = 8/3$ can be used to encrypt the image.

4 Working Algorithm

We have implemented the above mentioned algorithm in a python code, encrypting and decrypting a sample image signal.

The Python code for this algorithm is available along with the resultant image samples on the following Google Colab Notebook : Numerical Methods Project.ipynb

Python Code for the algorithm along with its explanation :

Encryption :

a. Importing libraries and Sample Image and formatting it :

```
import numpy as np
import math
from PIL import Image
import matplotlib.pyplot as plt
# Load your own input image (replace 'your_image.jpg' with your file path)
input_image = Image.open('/content/Sample_image_signal.jpg')
# Convert the input image to grayscale
input_image = input_image.convert('L')
# Convert the grayscale image to a NumPy array
input_image = np.array(input_image)
# Normalise the input image to the range (0, 1)
input_image = input_image / 255.0
# Define the size of the image (M and N)
M, N = input_image.shape
# Step 2: Convert the Image into a 1-D Vector
input_vector = input_image.flatten()
```

Here, we have first imported the image signal into our program and to make it processable, we are converting it to a numpy array with values in range (0,1) and flattening it to 1 dimension.

b. Image Splitting :

```
# Step 3: Split the Image into Two Vectors (P1 and P2)
half_length = M * N // 2
P1 = input_vector[:half_length]
P2 = input_vector[half_length:]
```

We split the input image into two halves for separate encryption.

c. Chaotic Map Generation and Scrambling :

We first set the parameters for the Lorenz system. Then we define the ODEs.

```
# Set parameters
sigma = 10.0
rho = 28.0
beta = 8.0 / 3.0
# Define the Lorenz system equations
def lorenz_system(t, xyz):
    x, y, z = xyz
    dxdt = sigma * (y - x)
    dydt = x * (rho - z) - y
    dzdt = x * y - beta * z
    return [dxdt, dydt, dzdt]
```

Having defined the Lorenz system, we solve the system using the Runge-Kutta Method of fourth order to obtain the Chaotic sequence. We scale the Chaotic sequence to match the size of our image signal.

```
# Runge-Kutta method implementation
def runge_kutta(f, t0, tf, y0, h):
    t_values = [t0]
    y_values = [y0]
```

```
    t = t0
    while t < tf:
        k1 = np.array(f(t, y_values[-1]))
        k2 = np.array(f(t + h/2, y_values[-1] + h/2 * k1))
        k3 = np.array(f(t + h/2, y_values[-1] + h/2 * k2))
        k4 = np.array(f(t + h, y_values[-1] + h * k3))
        y_next = y_values[-1] + (h / 6) * (k1 + 2*k2 + 2*k3 + k4)
        t += h
        t_values.append(t)
        y_values.append(y_next)
    return t_values, y_values
# Define the initial conditions and time span
initial_conditions = [1.0, 0.0, 0.0]
t0 = 0.0
tf = 1200.0
h = 0.01

# Solve the Lorenz system using the Runge-Kutta method
t_values, y_values = runge_kutta(lorenz_system, t0, tf,
initial_conditions, h)
# Extract the 'x' and 'y' variables
x_values = [y[0] for y in y_values]
y_values = [y[1] for y in y_values]
# Define Chaotic sequences
chaotic_sequence1 = x_values[:len(P1)]
chaotic_sequence2 = y_values[:len(P2)]
# Normalise chaotic sequences to (0, 1)
chaotic_sequence1 = np.array(chaotic_sequence1)
chaotic_sequence2 = np.array(chaotic_sequence2)
chaotic_sequence1 = (chaotic_sequence1 - chaotic_sequence1.min()) /
(chaotic_sequence1.max() - chaotic_sequence1.min())
chaotic_sequence2 = (chaotic_sequence2 - chaotic_sequence2.min()) /
(chaotic_sequence2.max() - chaotic_sequence2.min())
```

The obtained chaotic sequences have been stored in the specified variables. Using them, we will scramble the image vectors according to the random sequence present in the chaotic maps.

```
# Step 5: Scramble P1 and P2 with Chaotic Sequences
P1 = P1[np.argsort(chaotic_sequence1)]
P2 = P2[np.argsort(chaotic_sequence2)]
```

d. Recombining the two halves of the image :

```
combined_vector = np.concatenate((P1, P2))
```

e. Second layer of Encryption :

We generate a random security key and use the Bitwise XOR operation to add a second layer to the previously generated encryption.

```
# Generate the secret_key with the same size as combined_vector
secret_key = np.random.randint(0, 256, size=len(combined_vector),
dtype=np.uint8)
# Convert combined_vector to uint8
adjusted_vector_uint8 = (combined_vector * 255).astype(np.uint8)
# Perform bitwise XOR
diffused_vector = np.bitwise_xor(adjusted_vector_uint8, secret_key)
```

f. Finishing touch :

We reshape the obtained diffused vector to get the final Cipher Image, ready to be transmitted.

```
# Step 9: Reconstruct the Cipher Image
cipher_image = diffused_vector.reshape((M, N))
# Step 10: Save the Cipher Image (For demonstration, saving as a PNG)
cipher_image = (cipher_image).astype(np.uint8)
Image.fromarray(cipher_image).save("cipher_image.png")
```

Decryption :

For Decryption, one needs the same security key as generated earlier and the initial conditions for the chaotic sequence. Here, we will not be showing the code for decryption because it is not the objective of this report. The code can however be accessed on the above mentioned Colab Notebook. The steps involve the following:

a. Conversion of the cipher image to processable format.
b. Decrypting the second layer of encryption using the Bitwise XOR operator and the security key.
c. Generating the same chaotic sequences with the initial conditions available and using the Runge-Kutta method.
d. Splitting the cipher image into two halves and then descrambling them with the Inverse of the Chaotic sequence.
e. Recombining the obtained halves of the image signal and reshaping it into the image format. This image obtained is the Restored Image read from the encrypted signal.

5 Result and Discussions

A. Image Encryption

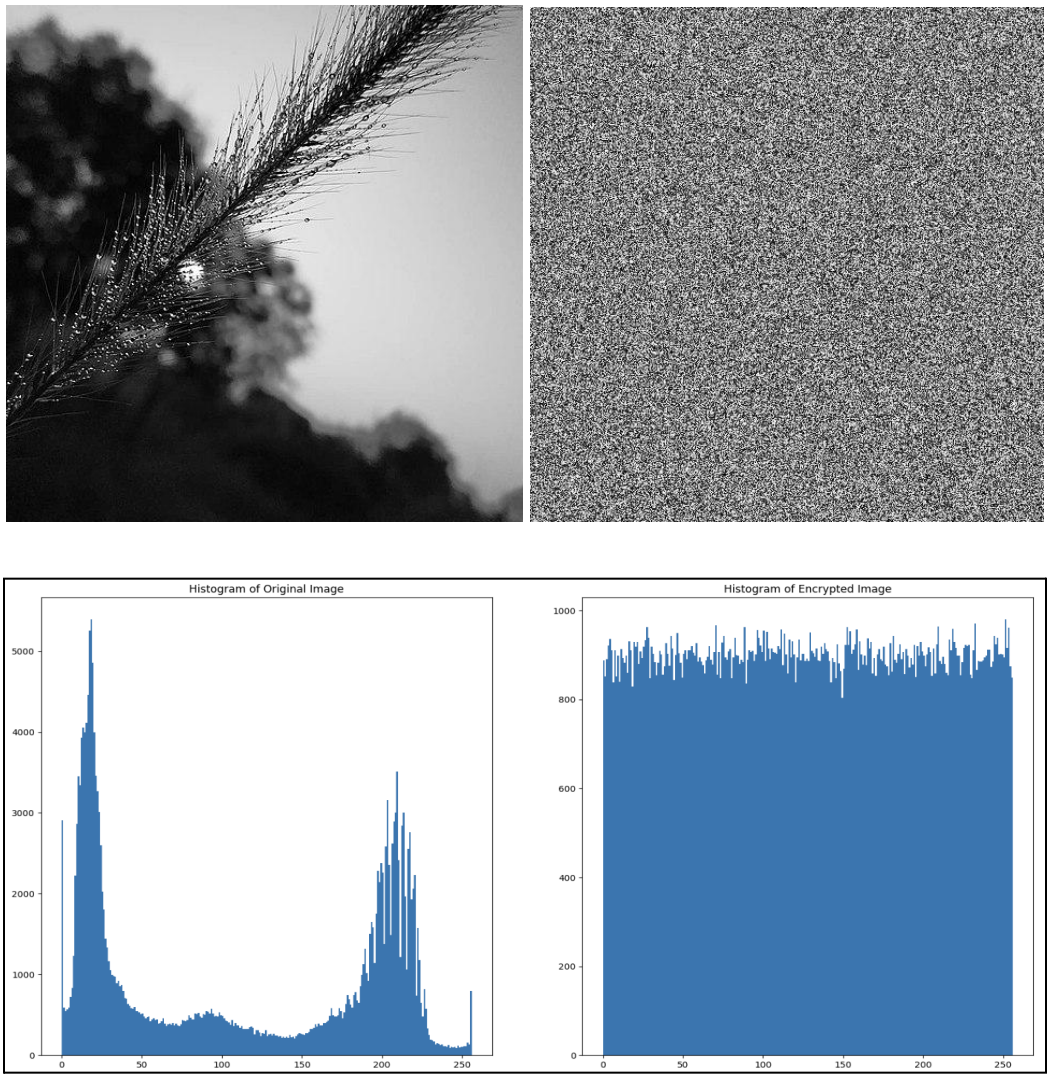


Fig 3. Input image with histogram, encrypted image with histogram

B. Decrypting Image

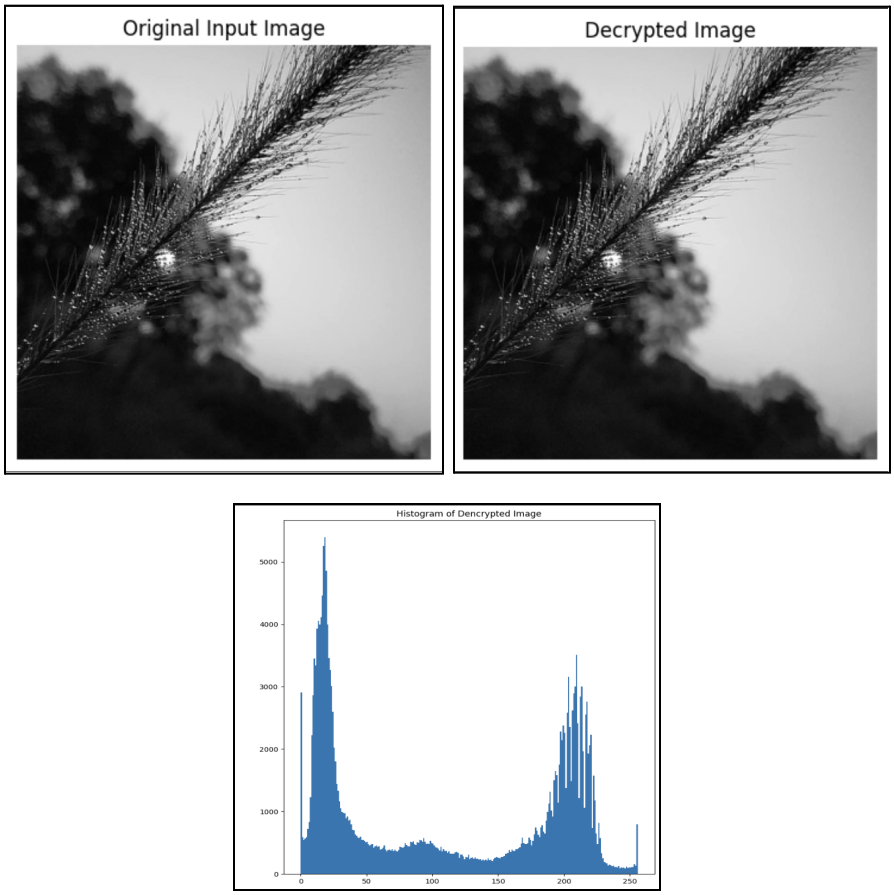


Fig 4. Input image , decrypted image with histogram

The histogram in the above images depict the frequency of each pixel value in specific image. The evenly distributed histogram of encrypted image signifies that every pixel is randomly scrambled and the encryption is secure. As outlined above, the nearer the distribution numbers that represent encrypted data, the better the encryption level is.

C. Parametric Measurements:

1. Structural Similarity Index (SSIM): This parameter is used to measure the pixel similarity between input image and the decrypted image.

```
import cv2
```

```
import matplotlib.pyplot as plt

import numpy as np

from skimage.metrics import structural_similarity as ssim

# Read the original and encrypted images

img_original = cv2.imread('/content/Black_and_white.jpg')

img_encrypted = cv2.imread('/content/uncipher_image.png')

# Calculate the SSIM values

ssim = ssim(img_original, img_encrypted, multichannel=True)

# Print the table

print('Parametric Measurements:')

print('SSIM:', ssim)
```

```
Parametric Measurements:
SSIM: 1.0
```

SSIR value for the original and decrypted image is 1.0, which indicates that our original and decrypted images are structurally similar.

In conclusion, the encryption and decryption of images using chaotic functions derived from the Lorenz system and solved through the Runge-Kutta method provide a robust and secure approach to safeguarding sensitive image data. The use of chaotic sequences as keys ensures a high level of confidentiality, making it suitable for various applications, including secure image transmission and storage. However, it's essential to manage and protect the secret key for successful decryption and to maintain the security of the system.

6 References

[1] Lorenz system of equations. Available online: Link

[2] DES encryptions method, Laissez Faire City Times, Vol 2, No. 28. Available : Link

[3] "Cryptography Introduction," GeeksforGeeks, 02-Nov-2018. [Online]. Available: Link

[4] "Chaos -- from Wolfram MathWorld," Wolfram MathWorld, 15-Mar-2006. [Online]. Available: Link