Report for ODE month-long Project

BTech (Information Technology and Mathematical Innovations), 1st Semester

Chaos-based image encryption using the Lorenz system

Agriya Khetarpal, Raunak Singh, Naman Priyadarshi, Saransh Chopra

Cluster Innovation Centre
University of Delhi

Certificate of Completion

This is to certify that the following persons: Agriya Khetarpal, Raunak Singh, Naman Priyadarshi, Saransh Chopra have completed this ODE project under my guidance and supervision as per the contentment of the requirements of the first semester in the course BTech (Information Technology and Mathematical Innovations) at the Cluster Innovation Centre, University of Delhi.

Dr Harendra Pal Singh
Assistant Professor
Department of Mathematics
Cluster Innovation Centre
University of Delhi

Acknowledgement

It is a great pleasure for us to present this project titled "*Chaos-based image encryption using the Lorenz system*" as a part of the first semester's curriculum in the BTech (Information Technology and Mathematical Innovations) course at the Cluster Innovation Centre, University of Delhi.

We would like to thank our teacher and mentor, Dr Harendra Pal Singh, who helped render this project an adequate success.

Abstract

Chaos theory is an interdisciplinary branch of mathematics and physics that deals with the mathematical concept of chaos described by dynamical systems – a function whose output in a three-dimensional geometrical plane depends upon the time variable. Many dynamical systems exist in nature, from the simple pendulum to the water flow in a pipe, irregularities in human heartbeat, or even the weather in a particular region. However, some dynamical systems exist whose motion is not governed by simple laws and is defined by specific non-periodic properties and high sensitivity to initial conditions. Unlike in non-chaotic systems, here, even infinitesimally small changes to the initial parameters yield a significantly different trajectory of the particle/entity in motion, limiting the system's predictability. The focus of our project is to encrypt and decrypt an image using the Lorenz chaotic system.

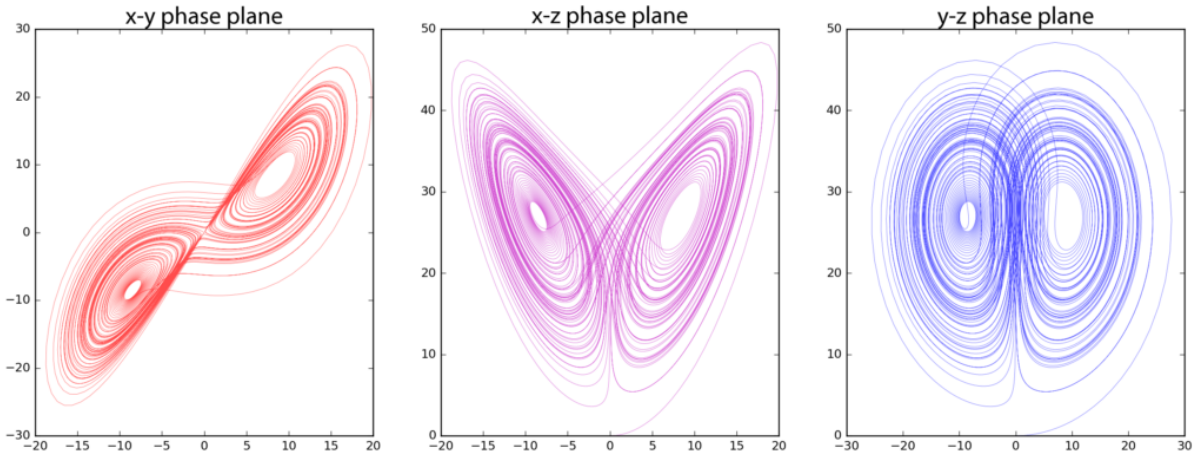
Index

Body

In this month-long project, our aim is to achieve the goal of encrypting an image using a chaotic system. The chaotic system used is the Lorenz set of linear ordinary differential equations, which are given by

$$\frac{dx}{dt} = \sigma(y - x),$$
$$\frac{dy}{dt} = x(\rho - z) - y,$$
$$\frac{dz}{dt} = xy - \beta z,$$

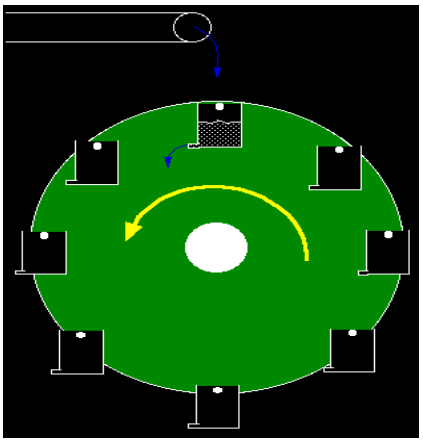
which attains the shape of a two fixed point attractor for the values $\rho = 28,\ \sigma = 10,$ and $\beta = \frac{8}{3}$



The Lorenz chaotic attractor was discovered by Edward Lorenz in 1963 when he was investigating a simplified model of atmospheric convection. It is a nonlinear system of three differential equations. With the most commonly used values of three parameters, there are two unstable critical points. The solutions remain bounded but orbit chaotically around these two points.

A physical model simulating the Lorenz equations has been attributed to Willem Malkus and Lou Howard around 1970. It consists of leaking cups on the rim of a larger wheel, as shown in the diagram on the right. Liquid flows from the pipe at the top; each cup leaks from the bottom.

Under different input flow rates, you should convince yourself that the wheel will spin one way under just the right flow rate and then the other chaotically.



The Lorenz system is deterministic, which means that if you know your variables' exact starting values, then, in theory, you can determine their future values as they change with time. Lorenz demonstrated that if you begin this model by choosing some values for x, y, and z, and then do it again with just *slightly* different values, you will quickly arrive at fundamentally different results. You can never know the *exact* value of any physical measurement in real life, although you can get close.

The Lorenz system is non-linear in nature with two existing non-linearities, i.e., xy and xz

Symmetry within the Lorenz system of differential equations also exists. The equations are invariant under the definition $(x,\ y) \rightarrow (-\ x,\ -\ y)$. Hence, if a point in the phase space $((x(t),\ y(t),\ z(t))$ is a solution, then it follows that the point $(-\ x(t),\ -\ y(t),\ z(t))$ is also a solution.

The point $(x^*, y^*, z^*) = (0, 0, 0)$ is a fixed point of the Lorenz system for all values of the parameters σ, ρ, and β. For $\beta > 1$, there is also a pair of points at $x^* = y^* = \sqrt{(\beta)(\rho - 1)}$ which coalesce with the origin point in a *pitchfork bifurcation* – a particular type of bifurcation where a system of differential equations transitions from having one fixed point to three fixed points.

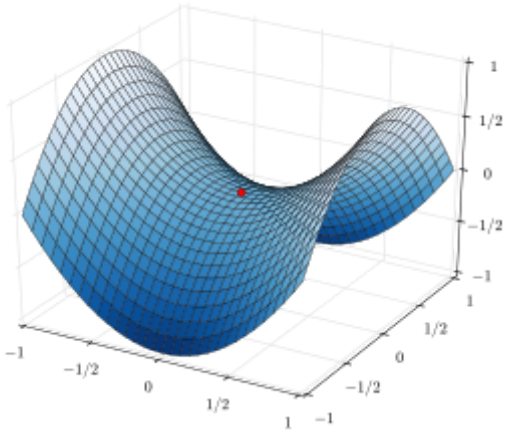
Stability analysis of the Lorenz system

Linearisation of the equations mentioned above yields
$\dot{x} = \sigma(y - z)$
$\dot{y} = \rho(x - y)$
$\dot{z} = -\beta z$

$$\begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix} = \begin{pmatrix} -\sigma & \sigma \\ \rho & -1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

The trace $\tau = -\sigma - 1 < 0$ and determinant $\triangle = \sigma(1 - \rho)$
For $\rho > 1$, origin is a saddle point since $\triangle < 0$

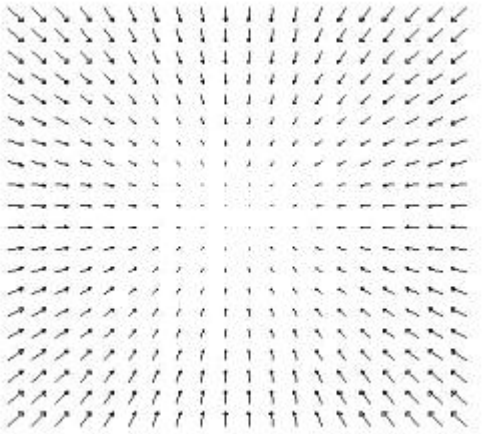
In differential equations, a *saddle point* or *minimax point* is a point on the surface of the graph of a function where the slopes (derivatives) in orthogonal directions are all zero (a critical point), but which is not a local extremum of the function. An example of a saddle point is when there is a critical point with a relative minimum along one axial direction (between peaks) and at a relative maximum along the crossing axis. However, a saddle point need not necessarily be in this form.



For example, in the above plotted graph $f(z) = x^2 - y^2$ the point (0,0) is the saddle point.

For $\rho < 1$, origin is a *sink* since $\tau^2 - 4\triangle = (\sigma + 1)^2 - 4\sigma(1 - \rho) = (\sigma - 1)^2 - 4\sigma\rho > 0$

A sink is defined as a point where all the local sets of points in the traced function seem to coalesce towards, it can be thought to be analogous to a regular kitchen sink installed in our homes and how the fluid entering the sink wishes to exit from one specific point.



Actually, for $\rho < 1$ it can be shown that every trajectory approaches the origin as $t \rightarrow \infty$ the origin is globally stable, hence there can be no limit cycles or chaos for $\rho < 1$

The substitution method

We attempt to describe the process used by us to achieve the goal of our project here. We first import all the essential libraries which will be necessary, such as *matplotlib* and *NumPy*. We then ask the user to enter the path of the image they wish to encrypt. For the sake of confirmation, we display the input image to the user. We then store the size of the image in the form of variables. We then generate a key for every pixel of the image in terms of its location and name it as the Lorenz key. This key is defined by the parameters we use in the Lorenz system, and hence, it can be easily customised and varied. We then initialise an empty image containing the new space for the encrypted form of the input image. Now we attempt to explain the central part of this project, the process of encryption. It is essential to note in this method; we are just trying to encrypt by substituting pixels. In another approach that we have attempted to explain after this, we shuffle and substitute the pixels to secure the image further. We now XOR each pixel with a pseudo-random number obtained using the pseudo-random number generator function created previously. We multiply each pseudo-random number with an enormous value of the order 10^5 and access the remainder generated by dividing the result by 256 to create random numbers between 0 and 255 (both inclusive). We then create the encrypted image by replacing it with the final pixels we get after subjecting the initial pixels to the math we described above. We then display the encrypted image. We initialise another empty image to store the decrypted image once we subject the encrypted one to the exact reverse procedure we followed to encrypt it and then display the decrypted image.

The shuffle and substitution method

We again start by importing most of the important libraries required to perform the project we aim to achieve, such as *matplotlib* and *NumPy.* We then accept the image entered by the user they wish to encrypt. Again, for the sake of confirmation, we display the entered image to the user. We again store the size of the image in the form of variables. We generate the same key as last time again by naming it as the Lorenz key and creating three lists. This key, again, is defined by the parameters we use in the Lorenz system and hence can be easily customised and varied. We now initialise empty index lists to store an index of the pixels of the image. We now initialise an empty image again to store the encrypted image to be generated. We now populate the generated lists with the image's dimensions, preparing them to undergo the encryption process. We proceed with the first step of encryption by shuffling the X index values using a simple conditional algorithm explained in the code. We do the same with the Y index, which has increased security by a greater level than before. We create an encrypted image using these shuffled indices and show the user the encrypted image after this step. This is where the shuffling part of our method is over. We now subject this encrypted image to the same substitution method we used in the previous section.

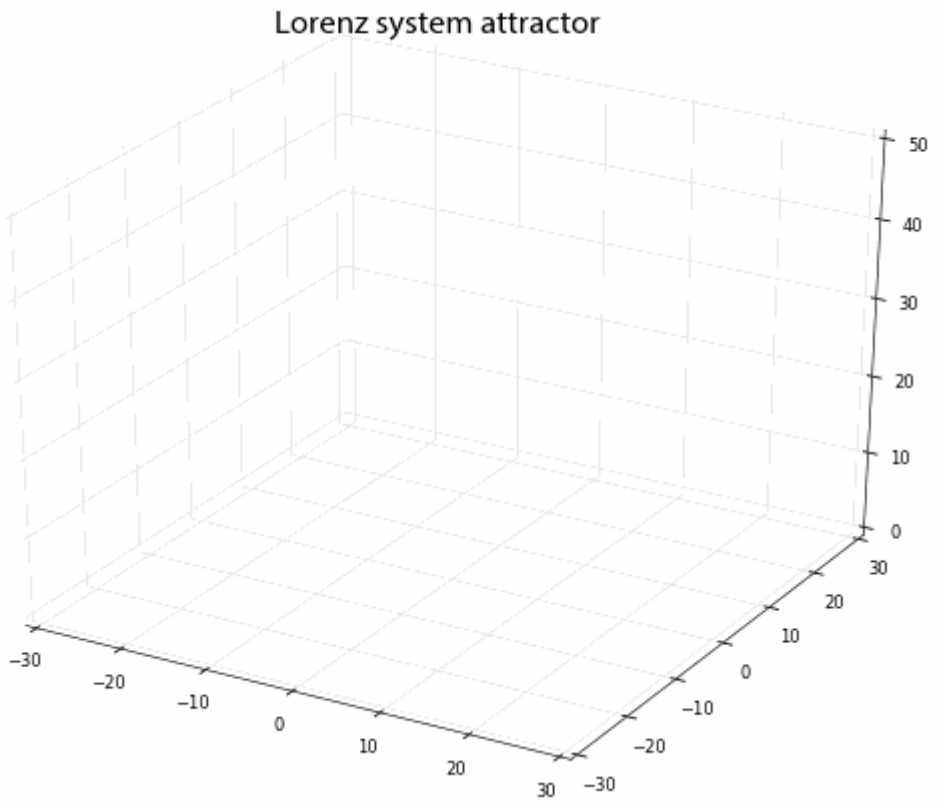
Confusion and diffusion

In cryptosystems, two desirable properties to hinder fast-scale decryption using statistical analysis or other methods are confusion and diffusion. Confusion in encryption refers to the process of encrypting an image wherein the data is changed drastically from the input to the output, i.e., making it challenging to find the key, increasing the ambiguity and vagueness of the data as a result.

Diffusion refers to the process of every part of the input data affecting every part of the output to render the analysis of the cryptosystem a more demanding task. In encryption, as opposed to confusion, diffusion is done to reduce the number of attempts to deduce the key in itself.

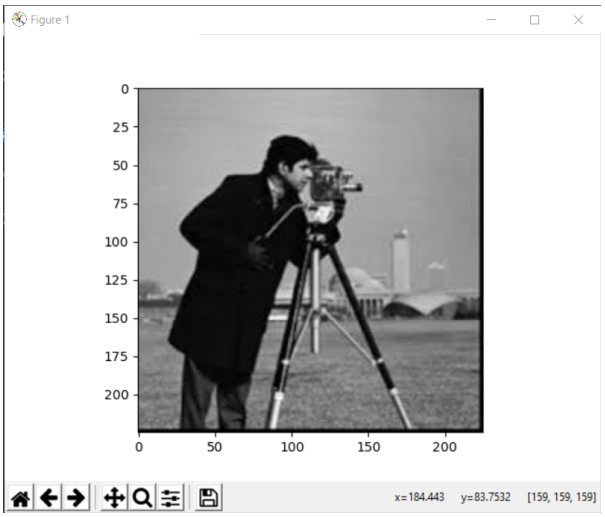
However, diffusion is not a hundred percent accurate and leaves some underlying patterns every time. If the diffusion process is fair, these patterns get scattered throughout the output. If multiple patterns exist, they are scattered amongst each other. Good diffusion processes and practices vastly increase the data required to analyse the cryptosystem and crack it.

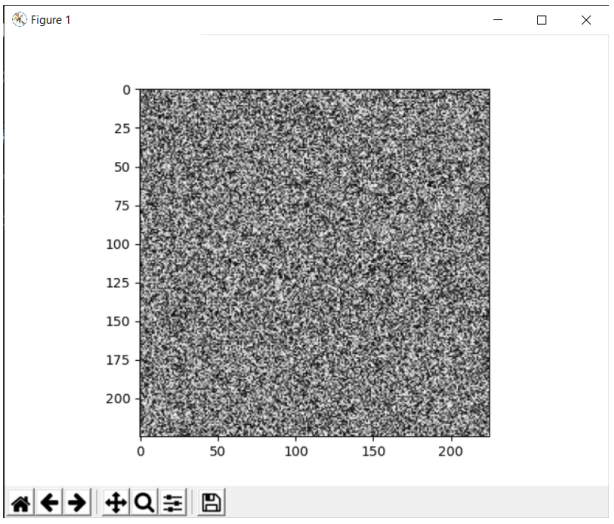
These two properties have a significant role to play in designing encryption methods and systems. A combination of good confusion and diffusion helps increase the security of a system. For example, the AES (Advanced Encryption Standard) specification has excellent properties.


Lorenz system attractor
50
40
30
20
10
0
30
20
10
0
−10
−20
−30
−30
−20
−10
0
10
20
30

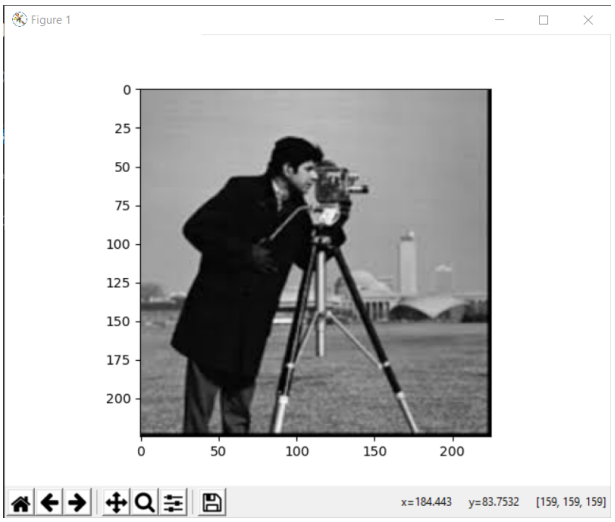
Results

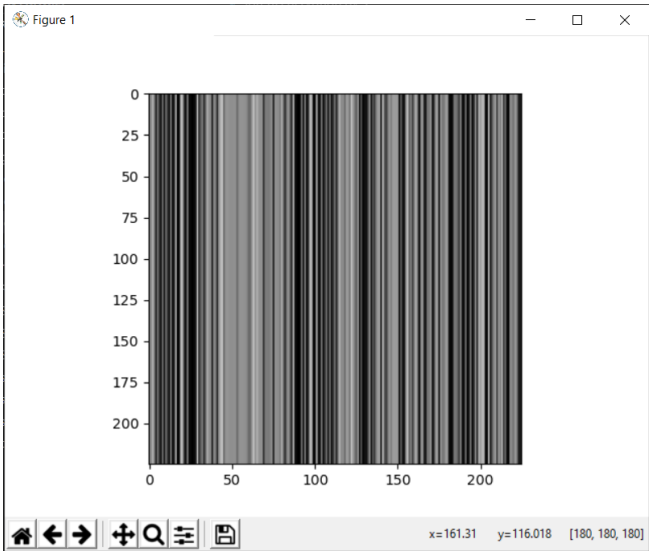
Substitution on a black-and-white image

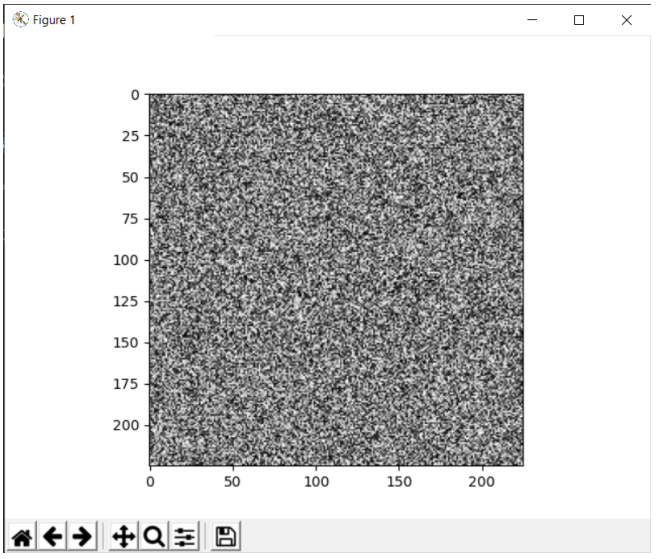




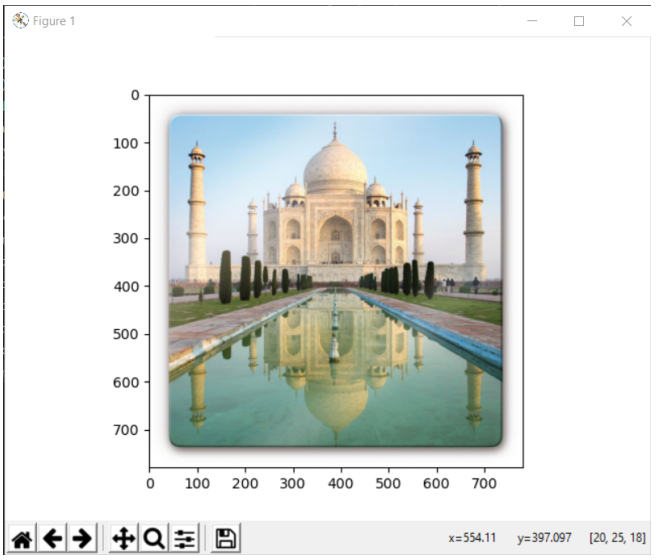
Shuffle and then substitution on a black-and-white image

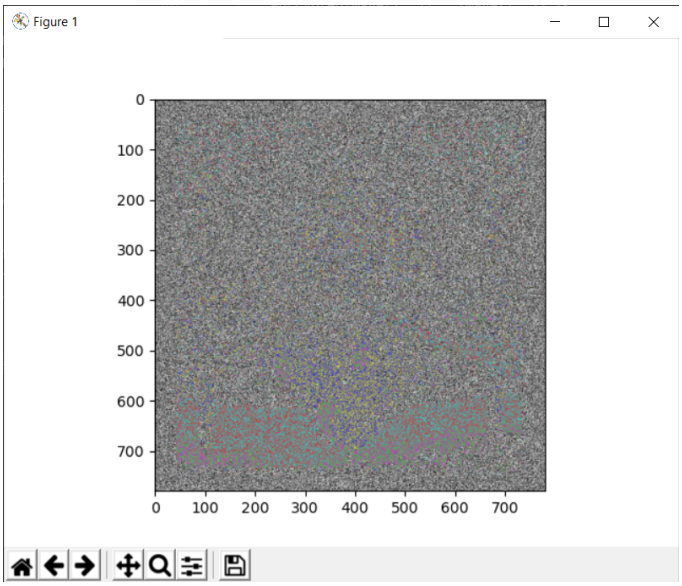




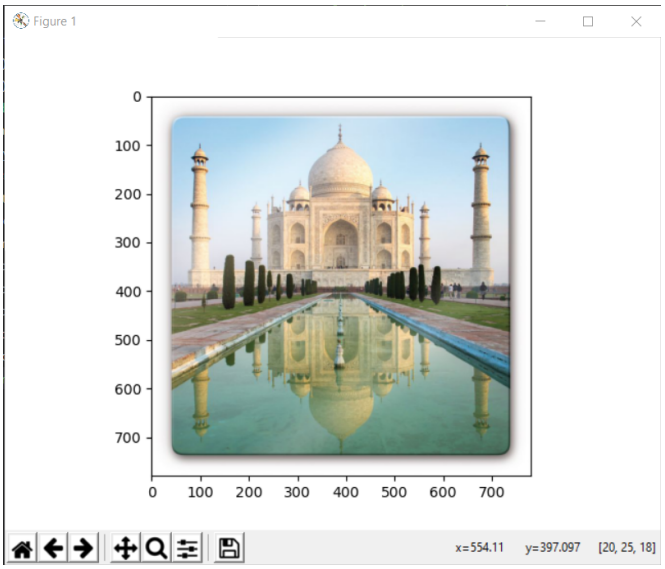


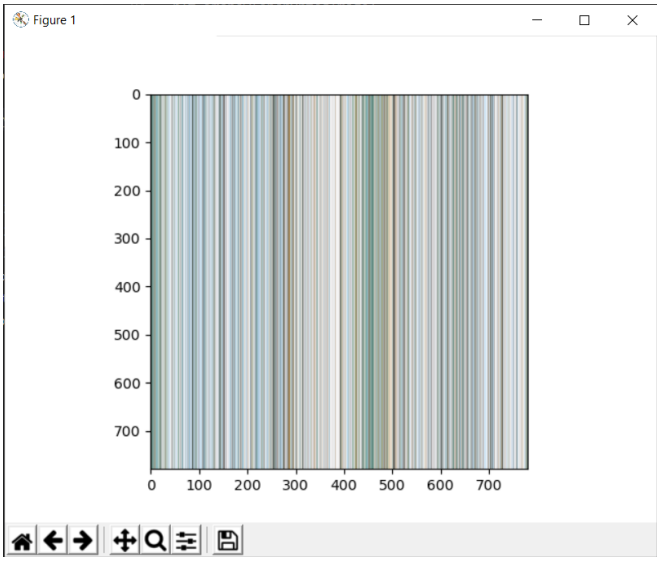
Substitution on a coloured image

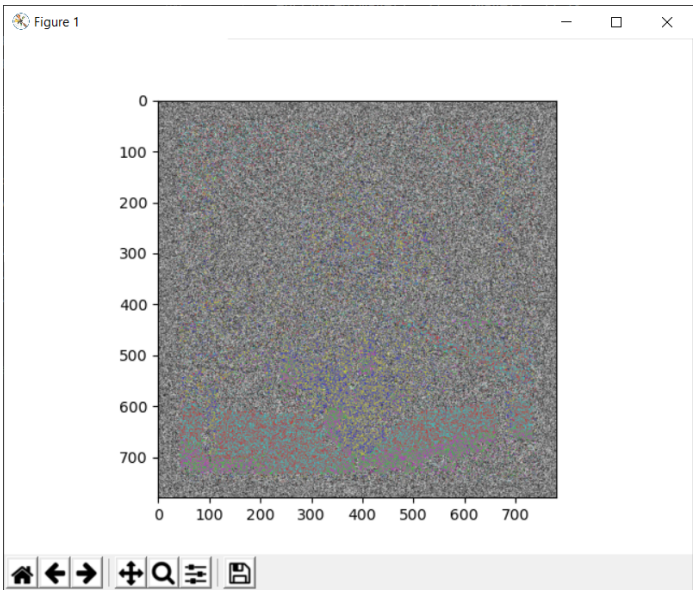




Shuffle and then substitution on a coloured image







Future Work

Based on the progress we have made so far in our effort towards this work, we have come up with certain ideas we aim to implement in the future regarding the same:

1. We aim to extend this method to video encryption since videos, essentially, are a collection of continuous frames, i.e. images. We can encrypt different frames with different systems so that the whole video can not be decrypted by one approach. The image frames can be shuffled in themselves as well, using an appropriate cipher to add an extra layer of security.

2. Dynamic encryption: the encrypted image can keep on changing with time; that is, the key to the image which governs the encryption process can be made dynamic (changing with time). Therefore, we will need to implement a process through which the repeatedly changing key can be stored for a particular instant to ease the decryption process while increasing security.

3. We can encrypt an image based on certain sectors classified by certain shapes and patterns inside it. For example, we can use object classification and other computer vision techniques to identify items such as a flower pot or a tiger and encrypt them using different paraphernalia compared to the surroundings. Otherwise, different parts of the image can be divided (into squares and rectangles), and the same process can be done on them.

4. We can encrypt the key to the system using similar encryption methods to ‘hide’ the Lorenz system’ and implement two or possibly higher (n) factor authentication.

5. Another topic of interest is enhancing the security of these encrypted systems in general by testing them against brute force

attacks and against AI-based attacks (for example, convolutional neural networks trained to solve jigsaw puzzles) and so on.

6. In the future, we can encrypt images using other chaotic and hyperchaotic systems, for example, the double and the triple pendulums, the Chua's circuit, the Rossler attractor, or even cellular neural networks.

References

1. Chen, Jun-xin & Zhu, Zhi-liang & Fu, Chong & Yu, Hai & Zhang, Li-bo. (2015). A fast chaos-based image encryption scheme with a dynamic state variables selection mechanism. Communications in Nonlinear Science and Numerical Simulation. 20. 846–860. 10.1016/j.cnsns.2014.06.032.

2. Somaya Al-Maadeed, Afnan Al-Ali, Turki Abdalla, "A New Chaos-Based Image-Encryption and Compression Algorithm", *Journal of Electrical and Computer Engineering*, vol. 2012, Article ID 179693, 11 pages, 2012. https://doi.org/10.1155/2012/179693

3. Shenyong Xiao, ZhiJun Yu, YaShuang Deng, "Design and Analysis of a Novel Chaos-Based Image Encryption Algorithm via Switch Control Mechanism", *Security and Communication Networks*, vol. 2020, Article ID 7913061, 12 pages, 2020. https://doi.org/10.1155/2020/7913061

4. Bhagat, A., Abhishek Surve, Sanuj Kalgutkar and Apeksha Waghmare. “Chaos Based Image Encryption and Decryption.” (2016).

5. https://arxiv.org/ftp/arxiv/papers/1211/1211.0090.pdf

6. Alvarez, Gonzalo & Li, Shujun. (2006). Some Basic Cryptographic Requirements for Chaos-Based Cryptosystems.. I. J. Bifurcation and Chaos. 16. 2129-2151. 10.1142/S0218127406015970.

7. Chaos-based Cryptography: Theory, Algorithms and Applications. Germany: Springer Berlin Heidelberg, 2011.

8. Gao, Tiegang & Chen, Zengqiang. (2008). Image encryption based on a new total shuffling algorithm. Chaos, Solitons & Fractals. 38.

213-220. 10.1016/j.chaos.2006.11.009.

9. https://youtu.be/DOwy0SuGD7Q

10. X. Zhang, L. Wang, Z. Zhou and Y. Niu, "A Chaos-Based Image Encryption Technique Utilizing Hilbert Curves and H-Fractals," in IEEE Access, vol. 7, pp. 74734-74746, 2019, doi: 10.1109/ACCESS.2019.2921309.

11. Rao, Surya. (2015). A New Chaotic Algorithm For Image Encryption And Decryption Of Digital Color Images. International Journal of Applied Engineering Research. 10.

12. Zhang, Jiangang & Zhang, Li & An, Xinlei & Luo, Hongwei & Yao, Kutorzi. (2016). Adaptive coupled synchronization among three coupled chaos systems and its application to secure communications. EURASIP Journal on Wireless Communications and Networking. 2016. 10.1186/s13638-016-0630-4.

13. Bonilla, L.L., Alvaro, M. & Carretero, M. Chaos-based true random number generators. *J.Math.Industry* 7, 1 (2016). https://doi.org/10.1186/s13362-016-0026-4

14. E. Solak, "Partial identification of Lorenz system and its application to key space reduction of chaotic cryptosystems," in IEEE Transactions on Circuits and Systems II: Express Briefs, vol. 51, no. 10, pp. 557-560, Oct. 2004, doi: 10.1109/TCSII.2004.834534.

15. Communication theory of secrecy systems," Bell Systems Technical Journal 28 (1949), 656 – 715

16. Lect6.dvi (ox.ac.uk)

Appendix

Lorenz key generator

```python
# Importing the required libraries
import numpy as np
import matplotlib.pyplot as plt

def lorenz_key(xinit, yinit, zinit, num_steps):
    """
    This function returns 3 lists of pseudo-random
    numbers generated using Lorenz system of differential
    equations.

    Parameters:
        xinit: float
            initial value of x
        yinit: float
            initial value of y
        zinit: float
            initial value of z
        num_steps: int
            number of keys required
            in a single list

    Returns:
        3 lists with pseudo-random numbers
        as their elements
    """

    # Initializing dt to a small value
    dt = 0.01

    # Initializing 3 empty lists
    xs = np.empty(num_steps + 1)
    ys = np.empty(num_steps + 1)
    zs = np.empty(num_steps + 1)

    # Initializing initial values
    xs[0], ys[0], zs[0] = (xinit, yinit, zinit)

    # Initializing constants
    s = 10
    r = 28
    b = 2.667

    # System of equations
    for i in range(num_steps):
        xs[i + 1] = xs[i] + (s * (ys[i] - xs[i])) * dt)
        ys[i + 1] = ys[i] + ((xs[i] * (r - zs[i])) - ys[i]) * dt)
        zs[i + 1] = zs[i] + ((xs[i] * ys[i] - b * zs[i]) * dt)

    # Uncomment to plot Lorenz system

    # fig = plt.figure()
    # ax = fig.gca(projection='3d')

    # ax.plot(xs, ys, zs)
    # plt.show()

    return xs, ys, zs

# Uncomment to plot Lorenz system

# lorenz_key(0.01, 0.02, 0.03, 1000)
```

Substitution algorithm

```
"""
Encrypting an image through substitution algorithm
using pseudo-random numbers generated from
Lorenz system of differential equations
"""

# Importing all the necessary libraries
import matplotlib.image as img
import matplotlib.pyplot as plt
import numpy as np
import lorenzSystem as key

# Accepting Image using it's path
path = str(input('Enter path of the image\n'))
image = img.imread(path)

# Displaying original image
plt.imshow(image)
plt.show()

# Storing the size of image in variables
height = image.shape[0]
width = image.shape[1]

# Using lorenz_key function to generate a key for every pixel
x, y, keys = key.lorenz_key(0.01, 0.02, 0.03, height*width)

l = 0

# Initializing an empty image to store the encrypted image
encryptedImage = np.zeros(shape=[height, width, 3], dtype=np.uint8)

# XORing each pixel with a pseudo-random number generated above/ Performing the
# substitution algorithm
for i in range(height):
    for j in range(width):
        # Converting the pseudo-random nuber generated into a number between 0 and 255
        zk = (int((keys[l]*pow(10, 5))%256))
        # Performing the XOR operation
        encryptedImage[i, j] = image[i, j]^zk
        l += 1

# Displaying the encrypted image
plt.imshow(encryptedImage)
plt.show()

# Initializing an empty image to store the decrypted image
decryptedImage = np.zeros(shape=[height, width, 3], dtype=np.uint8)

# XORing each pixel with the same number it was XORed above above/
# Performing the reverse substitution algorithm
l = 0
for i in range(height):
    for j in range(width):
        zk = (int((keys[l]*pow(10, 5))%256))
        decryptedImage[i, j] = encryptedImage[i, j]^zk
        l += 1

# Displaying the decrypted image
plt.imshow(decryptedImage)
plt.show()
```

Shuffle + Substitution algorithm

```
"""
Encrypting an image first by shuffling
it's pixels and then substituting the
shuffled pixels using pseudo-random
numbersngenerated from Lorenz system
of differential equations
"""

# Importing all the required libraries
import matplotlib.image as img
import matplotlib.pyplot as plt
import numpy as np
import lorenzSystem as key

# Accepting Image using it's path
path = str(input('Enter path of the image\n'))
image = img.imread(path)

# Displaying original image
plt.imshow(image)
plt.show()

# Storing the size of image in variables
height = image.shape[0]
width = image.shape[1]

# Using lorenz_key function to generate 3 lists of keys
xkey, ykey, zkey = key.lorenz_key(0.01, 0.02, 0.03, height*width)

# Initializing empty index lists to store index of pixels
xindex = []
yindex = []

# Initializing an empty image to store the encrypted image
encryptedImage = np.zeros(shape=[height, width, 3], dtype=np.uint8)
l = 0

# Populating xindex
for i in range(width):
    xindex.append(i)

# Populating yindex
for i in range(height):
    yindex.append(i)

# Re-arranging xindex and xkey to increase randomness
for i in range(width):
    for j in range(width):
        if xkey[i] > xkey[j]:
            xkey[i], xkey[j] = xkey[j], xkey[i]
            xindex[i], xindex[j] = xindex[j], xindex[i]

# Re-arranging yindex and ykey to increase randomness
for i in range(height):
    for j in range(height):
        if ykey[i] > ykey[j]:
            ykey[i], ykey[j] = ykey[j], ykey[i]
            yindex[i], yindex[j] = yindex[j], yindex[i]

# Shuffling original image's pixels and storing them
# in an empty image
for i in range(height):
    k = 0
    for j in range(width):
        encryptedImage[i][j] = image[yindex[k]][j]
        k += 1

# Shuffling original image's pixels and storing them
# in an empty image
for i in range(height):
    k = 0
    for j in range(width):
        encryptedImage[i][j] = encryptedImage[i][xindex[k]]
        k += 1

# Displaying the shuffled image
plt.imshow(encryptedImage)
plt.show()

# XORing each pixel with a pseudo-random number generated above/Performing the
# substitution algorithm
l = 0
for i in range(height):
    for j in range(width):
        # Converting the pseudo-random nuber generated into a number between 0 and 255
        zk = (int((zkey[l]*pow(10, 5))%256))
        # Performing the XOR operation
        encryptedImage[i, j] = encryptedImage[i, j]^zk
        l += 1

# Displaying the shuffled then substituted encrypted image
plt.imshow(encryptedImage)
plt.show()
```