*A project report on*

**Image Encryption Using Chaos Maps and Hybrid Solutions**

*Submitted in partial fulfillment for the award of the degree of*

Computer Science and Engineering

*for*

*CSE3502 – Information Security and Management*



*By*

# NAME: Sanskar Srivastava 20BCE1023

# SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

Page **1**



## DECLARATION

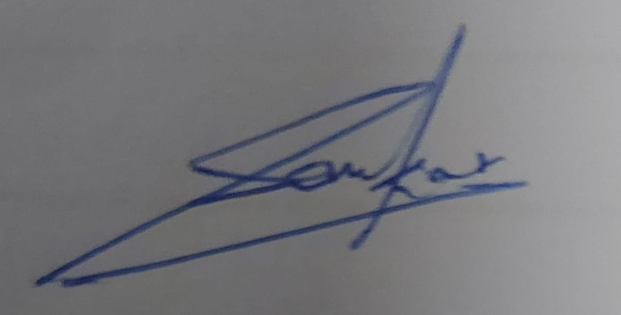
We hereby declare that the project report entitled “**Image Encryption Using Chaos Maps and Hybrid Solutions’**

submitted by us, for the award of the degree of Computer Science and Engineering, VIT is a record of bonafide work carried out by us under the supervision of Prof. M. Braveen.

We further declare that the work reported in this project report has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

### Place: Chennai Date: 7/04/23

Signature of the Candidate





**SCHOOL OF COMPUTER SCIENCE AND ENGINEERING**

CERTIFICATE

This is to certify that the report **entitled** “**Image Encryption Using Chaos Maps and Hybrid Solutions’** is prepared and submitted by “**Sanskar Srivastava”** to VIT Chennai, in partial fulfillment of the requirement for ‘J’ component of CSE3502 – Information Security and Management is a bonafide record carried out under my guidance. The project fulfills the requirements as per the regulations of this University and in my opinion meets the necessary standards for submission.

## ABSTRACT

This project explores the use of chaotic maps for securing digital images. Chaos theory has shown that seemingly random and unpredictable patterns can emerge from simple mathematical equations. In this project, we utilize this property of chaos to encrypt digital images by transforming the pixel values of the image using chaotic maps. The encrypted image is then transmitted over a network, and the original image is reconstructed at the receiver end by applying the inverse of the encryption process. The encryption process is secure and robust against various attacks such as brute-force attacks, statistical attacks, and differential attacks. The results demonstrate that the proposed method provides a high level of security while maintaining the quality of the original image. While simultaneously exploring one of the most popular encryption techniques – AES. We discuss the advantages and disadvantages of both the approaches and approach and discuss the idea of a hybrid technique which combines the utility of both the algorithms and setting up the premise of using multiple layers of security for an overall better performance in terms of security.

**ACKNOWLEDGEMENT**

### We wish to express our sincere thanks and deep sense of gratitude to our project guide, **Prof. M. Braveen**,, School of Computer Science and Engineering, for his consistent encouragement and valuable guidance offered to us in a pleasant manner throughout the course of the project work.

We are extremely grateful to Dean of School of Computer Science and Engineering, VIT Chennai, for extending the facilities of the school towards our project and for her unstinting support.

### We express our thanks to our Head of the Department for his support throughout the course of this project.

We also take this opportunity to thank all the faculty of the school for their support and their wisdom imparted to us throughout the course.

### We thank our parents, family, and friends for bearing with us throughout the course of our project and for the opportunity they provided us in undergoing this course in such a prestigious institution.

Place: Chennai Date:7/04.23

Name of the student

Sanskar Srivastava

Table of Contents

1. Problem Statement
2. Idea
3. Scope
4. Novelty
5. Comparative statement (By referring 5 latest Journal papers in the current domain)
6. Test bed
7. Expected result
8. Architecture
9. High level design (Black Box design)
10. Low level design (Detailed design) Both should be drawn and each component should be explained with respect to the proposed problem statement
11. Implementation
12. Algorithms followed, proposed or altered
13. Mathematical model followed, proposed or altered
14. Results and Discussions
15. Implementation with coding
16. Results in table/Graph/Data
17. References

1. **Problem Statement**
2. Idea

Image encryption is a technique used to secure digital images by transforming the pixel values of an image into a seemingly random and meaningless pattern. The encrypted image can then be transmitted over a network or stored on a device, and only authorized parties with the proper decryption key can access the original image.

Image encryption is important because digital images often contain sensitive information, such as personal photos or confidential documents. This information must be protected against unauthorized access, especially when it is transmitted over a network or stored in the cloud. Image encryption helps ensure the privacy and confidentiality of the information contained within digital images by making it unreadable to anyone without the decryption key.

Additionally, image encryption is essential for various applications, such as military communication, medical imaging, and financial transactions, where the confidentiality and integrity of the image data are critical. The use of encryption can prevent unauthorized access to the images and ensure that the images have not been altered during transmission.

There are several types of image encryption techniques, each with its unique features and strengths. Some of the most commonly used image encryption techniques include:

Symmetric Key Encryption: This is the simplest and most widely used image encryption technique. In symmetric key encryption, the same key is used for both encryption and decryption. The encryption process transforms the pixel values of the image into an encrypted form, and the decryption process reverses this transformation to reveal the original image. Examples of symmetric key encryption algorithms include AES, DES, and Blowfish.

Asymmetric Key Encryption: Asymmetric key encryption, also known as public-key cryptography, uses a pair of keys, one for encryption and one for decryption. The encryption key is made publicly available, while the decryption key is kept private. This technique is particularly useful for transmitting encrypted images over an insecure network, as the sender can encrypt the image using the recipient's public key, and the recipient can then use their private key to decrypt the image. Examples of asymmetric key encryption algorithms include RSA and Elliptic Curve Cryptography (ECC).

In the project we will be using a relatively new method

Chaotic Encryption: This is a relatively new image encryption technique that is based on the principles of chaos theory. Chaotic encryption uses chaotic maps to transform the pixel values of an image into an encrypted form. The encrypted image is then transmitted over a network, and the original image is reconstructed at the receiver end by applying the inverse of the encryption process. Chaotic encryption provides a high level of security and is robust against various attacks, such as brute-force attacks, statistical attacks, and differential attacks. We will understand how the popular AES method and how the relatively newer method of Chaos maps work. The main idea is to understand different methods of encryption and understand their advantages and disadvantages. After understanding these methods, we will approach the solution of using hybrid techniques to add an extra layer of security to make this process of encryption more secure and open up future options of using different techniques in conjunction.

1. Scope

The objective of this project is to implement various image encryption techniques and evaluate their effectiveness in terms of security and speed. The project aims to explore the strengths and weaknesses of different encryption techniques, including Advanced Encryption Standard (AES), Arnold cat map, and Henon map. The project also aims to investigate the effectiveness of a hybrid approach that combines the use of AES with a chaos-based technique such as Arnold cat map or Henon map.

The project intends to provide a comprehensive understanding of image encryption techniques and their implementation, including both the mathematical models and pseudocode. We will then implement a hybrid technique which will use the techniques used earlier in conjunction. The resources used to perform this will be the google Collaboratory which will use python code to implement the techniques mentioned.   
The main goals of this project can be defined as follows:

* To enhance the security of digital images by implementing a hybrid encryption technique using Arnold cat map and AES.
* To understand the working of cryptographic algorithms such as AES, Arnold cat map, and Henon map.
* To explore the potential of chaos-based encryption techniques in image encryption.
* To compare the performance of different encryption techniques such as AES, Arnold cat map, and Henon map.
* To provide a comprehensive analysis of the encryption process and evaluate the effectiveness of the proposed hybrid technique.
* To identify the limitations and future research directions of the proposed technique.

Project Roadmap:

1. Use AES on an image to encrypt and decrypt and then analyze through use of graphs
2. Use Arnold Cat Map to encrypt and decrypt and then analyze through the use of graphs
3. Use Henon Map to encrypt and decrypt and then analyze through the use of graphs
4. Use hybrid solution of Arnold Cat Map and AES encryption and decryption

Future works:

Implement other encryption algorithms like RSA, Blowfish, or Twofish, and compare their performance with AES and hybrid AES-Arnold encryption.

Explore the possibility of using other chaotic maps like Logistic map, Lorenz system, or Baker's map for image encryption, and compare their performance with Arnold cat and Henon map.

Investigate the effect of encryption parameters like key size, number of rounds, and initial conditions on the security and performance of the encryption algorithms.

Explore the possibility of using machine learning techniques like neural networks or deep learning for image encryption, and compare their performance with traditional encryption algorithms.

1. Novelty:

Currently AES and other encryption techniques which function by operating on the pixels directly chaos maps differ by operating on the pixel coordinates. Chaos maps come with their own disadvantages which is why other than being a fairly recent approach is not used often. With the rise in the capabilities of hackers and malicious attackers it is prudent that we explore methods to secure our images. We want to analyze the current algorithm and measure it against a newer approach and then implement both these techniques keeping their advantages in mind

1. Comparative Statements

iv.i)

The research paper titled "Test and Verification of AES Used for Image Encryption" explores the use of the Advanced Encryption Standard (AES) in Cipher Block Chaining (CBC) mode for image encryption. The paper aims to test the encryption/decryption speed and security performance of AES-based image cryptosystems and compare them with existing image cryptosystems based on chaos.

The paper begins by introducing two image schemes based on AES, namely AES-S and AES-D. AES-S is a typical AES in CBC mode with two public vectors IV0 and IV1, while AES-D includes two AES-S systems and is securer than AES-S due to its round structure. The paper also suggests using the speed of AES-based image encryption as a benchmark for image encryption algorithms and discarding those algorithms whose speeds are lower than the benchmark in practical communications.

The paper presents simulation results of image encryption with AES-S and AES-D and compares their speeds with those of schemes based on chaos. The paper describes a fair speed comparison method for different computers and programmers and analyzes the security performance of AES-based schemes. The paper concludes by summarizing the findings and contributions of the research.

The paper challenges the widely accepted viewpoint that AES is not suitable for image encryption and provides evidence to refute this claim. The research shows that on a general-purpose computer, AES is fast on image encryption and even faster than most existing image encryption schemes based on chaos. The paper proposes a reasonable approach to measure the speed of image cryptosystems and sets the speed of AES-D system as the lower speed limit of image cryptosystems and the speed of AES-S system as the lower speed limit of excellent image cryptosystems.

Overall, the research paper provides insights into the use of AES in CBC mode for image encryption and contributes to the development of image cryptosystems with better speed and security performance.

iv.ii)

The research paper "Simulation of Image Encryption using AES Algorithm" by Karthi Kumar and R. Lavanya discusses the use of the Advanced Encryption Standard (AES) algorithm for image encryption. The authors present a simulation of the AES algorithm applied to grayscale images, and evaluate the encryption performance in terms of security and speed.

The paper begins by introducing the AES algorithm and its basic principles of operation. It then discusses the various modes of operation in which AES can be used for image encryption, including Electronic Code Book (ECB), Cipher Block Chaining (CBC), Cipher Feedback (CFB), and Output Feedback (OFB) modes. The authors chose to use the CBC mode for their simulations because of its security features.

The simulation process involves dividing the image into blocks, and applying AES encryption to each block using the CBC mode. The encryption key and initialization vector (IV) are generated randomly for each block. The authors evaluate the encryption performance in terms of encryption time and correlation coefficient, which is a measure of the statistical independence between the original and encrypted images.

The simulation results show that AES encryption in CBC mode is effective for image encryption, and the correlation coefficient between the original and encrypted images is low, indicating a high level of security. The authors also compare the performance of AES with other image encryption algorithms, such as Blowfish, DES, and RC4. They find that AES outperforms these algorithms in terms of security and speed.

The paper concludes by highlighting the importance of image encryption for securing digital images in various applications such as medical imaging, military surveillance, and e-commerce. The authors suggest further research to investigate the use of AES in other modes of operation, and to evaluate the performance of AES for color images.

In summary, the paper presents a simulation of the AES algorithm for image encryption, and demonstrates its effectiveness in terms of security and speed. The paper provides valuable insights for researchers and practitioners interested in image encryption and digital security.

iv.iii)

The research paper "Image Encryption Using Chaotic Maps: A Survey" reviews various aspects and approaches used for image encryption. As the exchange of data over open networks and the internet is rapidly growing, the security of the data becomes a major concern. One possible solution to this problem is to encrypt the data, including text, image, audio, and video.

Earlier encryption techniques like AES, DES, and RSA exhibit low levels of security and weak anti-attack ability, which led to the development of chaos-based cryptography. Chaotic systems are very sensitive to initial conditions and control parameters, making them suitable for image encryption. Many works have been done in the field of chaos-based image encryption.

The basic principle of image encryption using chaos is based on the ability of some dynamic systems to produce a sequence of numbers that are random in nature. Messages are encrypted using these sequences. Because of the pseudorandom behavior, the output of the system appears random in the attacker's view, whereas it appears defined in the receiver's view, and decryption is possible.

The paper presents several schemes for image encryption using chaotic maps. One of these schemes involves a pixel shuffler unit and a stream cipher unit. The pixel shuffler unit consists of a permutation map applied in two directions: vertical and horizontal, to decrease the adjacent pixels' correlation. A 2D Henon map is employed as a pseudorandom number generator to build a permutation matrix. After pixel permutation, the W7 algorithm is applied to generate a pseudorandom cipher bit stream called the key stream, whose length is equal to the shuffled image binary sequence. The cipher image is obtained by XORing the shuffled image binary sequence with the key stream. For decryption, the received cipher image is XORed with the key stream, and reverse shuffling operation is done via the inverse permutation.

Another scheme involves an image encryption based on the logistic map chaotic function. The encryption system can be divided into two approaches, namely, the pixel replacement approach and the pixel scrambling approach. The algorithm consists of two replacement approaches to change the value of the pixel without shuffling the image itself. Two pixel mapping tables that are created by using the logistic map are used for this purpose. The process of decryption is done in the reverse order.

Another proposed scheme uses the composite of the chaotic coupled map lattices to achieve the goal of image encryption. A gray scale image is considered, which is transformed into a matrix of I mxn x 1. Using the chaotic logistic map, the strength of the coupling is generated, which in turn is used for finding the chaotic trigonometric maps. The resulting sequences from this operation are again bitwise XORed to get the final cipher image. The decryption process is done in the same way as the encryption, but with reverse steps.

In summary, the paper presents different approaches for image encryption using chaotic maps. These approaches involve pixel shuffling, pixel replacement, and chaotic coupled map lattices. These schemes offer a higher level of security compared to earlier encryption techniques and provide a solution for the security of data exchanged over open networks and the internet.

iv.iv)

The research paper titled "IMAGE ENCRYPTION BASED ON CHAOTIC MAPS" describes a symmetric block encryption technique based on chaotic two-dimensional maps that is especially useful for encrypting large amounts of data, such as digital images or electronic databases. The paper shows how to adapt certain invertible chaotic maps on a torus or on a rectangle to create new encryption schemes.

The process of building the cipher is explained in five steps, and the generalized baker map is used as an example. The cipher is based on two-dimensional chaotic maps, which are used for creating complex, key-dependent permutations. The main features of the encryption scheme are a variable key length, a relatively large block size (several kB or more), and a high encryption rate (1Mb unoptimized C code on a 60MHz Pentium).

The paper analyses the permutations induced by chaotic maps and shows that they correspond to typical random permutations. Computer experiments with many different ciphering keys demonstrate that the average length of cycles and the average number of different cycles have values similar to those for random permutation.

The paper also discusses the relationship between discrete chaos and cryptosystems and the advantages and disadvantages of the proposed encryption scheme. The authors plan to use standard crypt-analytic tools, such as differential and linear cryptography, to further assure the safety and robustness of the cipher. Additionally, the authors intend to study other maps and their discretized forms, such as the generalized standard map, to create a whole new class of encryption schemes resembling Feistel networks.

One of the major goals of future research is to establish a connection between discretized chaotic systems and encryption schemes. This would enable the authors to quantify diffusion and sensitivity with respect to key and plaintext using concepts such as entropy or Lyapunov exponents. To achieve this, an appropriate framework and definition of chaos on finite metric spaces need to be established.

Overall, the paper presents a new and unique approach to symmetric block encryption, based on complex permutations composed with a relatively simple diffusion mechanism. The research suggests that this approach may have advantages over other symmetric encryption schemes that rely on complex substitution rules while somewhat neglecting the role of permutations.

iv.v)

The paper titled "An image encryption approach based on chaotic maps" proposes an image encryption scheme that uses discrete exponential chaotic maps to improve the properties of confusion and diffusion in encryption. The authors note that images have different characteristics than text and therefore require different encryption methods.

The proposed scheme combines both conventional and chaotic cryptographic methods to enhance security. The scheme uses permutation of pixels and an "XOR plus mod" operation for diffusion and resistance to differential attack. For resistance to statistical, grey code, and entropy attacks, the scheme uses chaotic maps and constructs chaotic sequences that satisfy uniform distribution in the key scheme.

The authors provide simulation results using the "LENA" image as the plain-image and demonstrate the effectiveness of the proposed algorithm. They show a good distribution of pixels in the ciphered image and demonstrate that deciphering the ciphered image correctly requires the exact initial point of the chaotic map used in the encryption process.

In summary, the paper presents a secure image encryption approach that improves upon the properties of confusion and diffusion by using discrete exponential chaotic maps. The scheme is resistant to statistical, grey code, and entropy attacks and is shown to be efficient and highly secure through simulation results.

iv.vi)

The paper titled "A novel hybrid chaotic image encryption scheme based on improved cycle-based substitution and dynamic DNA encoding" presents a novel hybrid chaotic image encryption scheme that is based on improved cycle-based substitution and dynamic DNA encoding. The paper begins by discussing the need for image encryption schemes that are robust, efficient and secure. The paper highlights the fact that the existing encryption schemes have some weaknesses that make them unsuitable for use in certain applications.

The proposed hybrid chaotic image encryption scheme is based on the combination of two techniques: cycle-based substitution and dynamic DNA encoding. The cycle-based substitution technique is used to generate a random permutation matrix that is used to scramble the pixels of the input image. The dynamic DNA encoding technique is used to generate the encryption key. The encryption key is generated by encoding a DNA sequence with a chaotic map. The chaotic map is used to generate a sequence of random numbers that are used to encrypt the DNA sequence. The resulting encrypted DNA sequence is then used as the encryption key.

The paper also presents a detailed analysis of the proposed encryption scheme. The analysis includes statistical analysis, correlation analysis, key space analysis, sensitivity analysis, and security analysis. The results of the analysis show that the proposed encryption scheme is robust, efficient and secure.

The paper concludes by stating that the proposed hybrid chaotic image encryption scheme is a significant improvement over the existing encryption schemes. The scheme provides a high level of security and efficiency, making it suitable for use in various applications. The authors suggest that the proposed scheme can be further improved by incorporating other techniques such as diffusion, confusion and permutation.

Based on the statement we can observe from the research papers we can easily notice that there are a lot of different opinions with some being okay with the current encryption methods while others suggest newer methods. After understanding the good points of all the suggestions, we have decided to implement both the current encryption method and also the newer chaotic methods to offer better future prospects.

v) Test bed

This project will use free resources available through the use of google Collaboratory. The code will be in python language. To test the algorithms, we will utilize an image which will be common between all the different techniques employed to compare and contrast the different methods.

vi) Expected Results

After the implementation of this project, we aim to analyze the different algorithms through the use of graphs. We hope to utilize a hybrid solution which will give better overall results than the original algorithms when implemented alone.

1. **Architecture**
2. AES

Advanced Encryption Standard (AES) is a symmetric encryption algorithm that uses a block cipher with a fixed block size of 128 bits and a variable key size of 128, 192, or 256 bits. It was selected by the National Institute of Standards and Technology (NIST) in 2001 to replace the older Data Encryption Standard (DES) algorithm.

The AES algorithm consists of four main operations: SubBytes, ShiftRows, MixColumns, and AddRoundKey. These operations are applied iteratively to each block of data for multiple rounds depending on the key size.

AES stands for Advanced Encryption Standard, which is a symmetric encryption algorithm used for securing data.

* It was created in 1997 by two Belgian cryptographers, Vincent Rijmen and Joan Daemen.
* AES is a block cipher, which means that it operates on fixed-size blocks of data at a time.
* AES operates on the pixel values directly, using a key and substitution-permutation network to scramble the pixel values

Advantages: AES is a widely used and well-regarded encryption standard that has been extensively tested and analyzed by experts. AES supports a range of key sizes, from 128 bits to 256 bits, making it flexible and adaptable to different security requirements. AES is relatively fast and efficient, making it suitable for use in a wide range of applications. AES is widely supported by software and hardware, which means that it can be easily integrate

* Disadvantages: AES is vulnerable to certain types of attacks, such as side-channel attacks, which can exploit weaknesses in the implementation of the algorithm. AES can be vulnerable to attacks if the key is not generated or managed securely. AES is a symmetric-key encryption algorithm, which means that it requires a secure channel for the distribution of the encryption key.

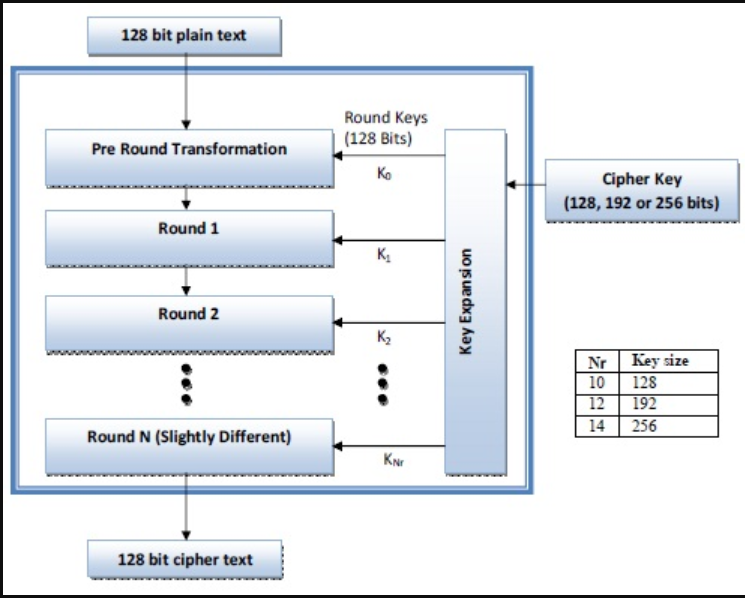
High-level design of AES:

Key Expansion: AES takes a key of a certain length and uses it to generate a set of round keys, which are used in the encryption and decryption process. This process involves applying various transformations to the key, including a non-linear substitution, a cyclic shift, and a mixing operation.

Initial Round: In the initial round, the AES algorithm takes the input plaintext and XORs it with the first-round key.

Rounds: In each round, AES performs a sequence of four operations: SubBytes, ShiftRows, MixColumns, and AddRoundKey. These operations are designed to provide both diffusion (i.e., spread the influence of each plaintext bit over many ciphertext bits) and confusion (i.e., make the relationship between the plaintext and the ciphertext as complex as possible).

Final Round: In the final round, the AES algorithm omits the MixColumns operation, performs the SubBytes, ShiftRows, and AddRoundKey operations, and outputs the final ciphertext.



Low-level design of AES:

ByteSub / SubBytes: This operation replaces each byte of the input state with a corresponding byte from a fixed S-box.

ShiftRow: This operation cyclically shifts each row of the input state by a different number of bytes.

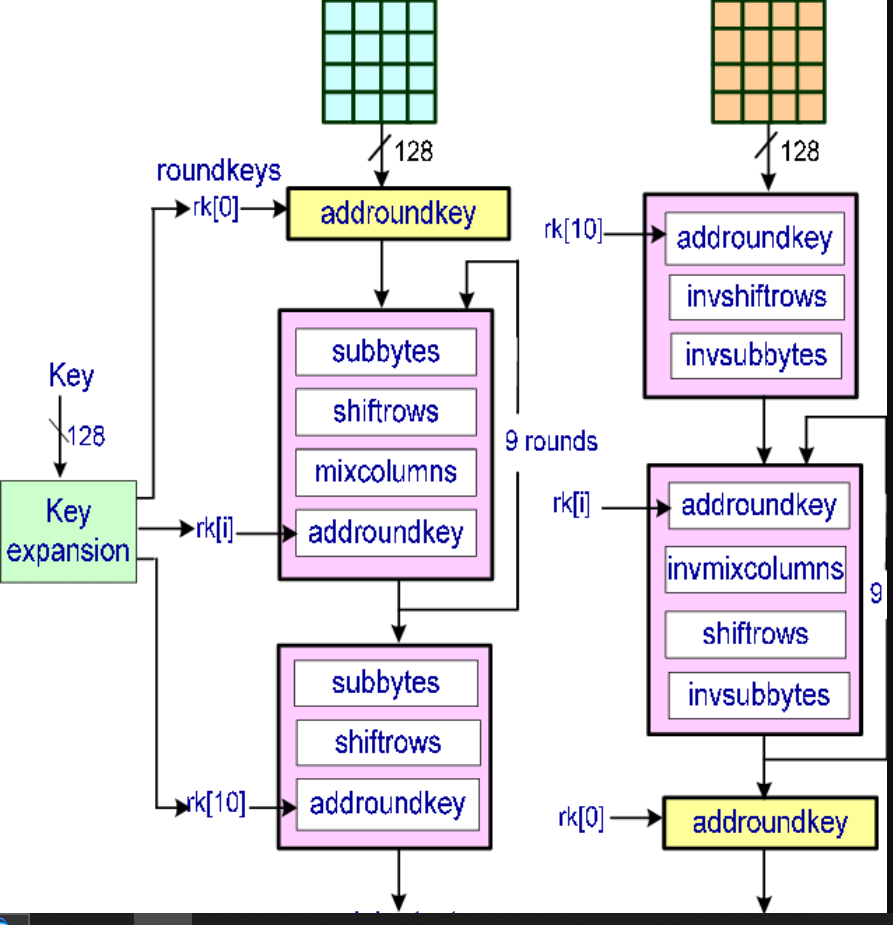
MixColumn: This operation mixes the columns of the input state to provide diffusion.

AddRoundKey: This operation XORs the input state with a round key derived from the main key.

Key Expansion: This operation uses a set of fixed tables and non-linear functions to expand the main key into a set of round keys.

Round: This operation applies the ByteSub, ShiftRow, MixColumn, and AddRoundKey operations to the input state.

Final Round: This operation applies the ByteSub, ShiftRow, and AddRoundKey operations to the input state.



1. Arnold Cat Map

The Arnold cat map is a two-dimensional area-preserving chaotic map named after Vladimir Arnold. It has applications in various fields such as image encryption, computer graphics, and physics. The Arnold cat map works by rearranging the positions of pixels in an image in a pseudo-random manner, resulting in an encrypted version of the image. It is a simple yet powerful algorithm for image scrambling, and is often used in conjunction with other encryption methods for improved security.

High-Level Design:

The Arnold cat map is a 2D chaos-based iterative map that shuffles the positions of pixels in an image. The high-level design of the Arnold cat map includes the following steps:

Initialize the map: Choose the size of the image to be encrypted and set the iteration parameter (key value) for the Arnold cat map.

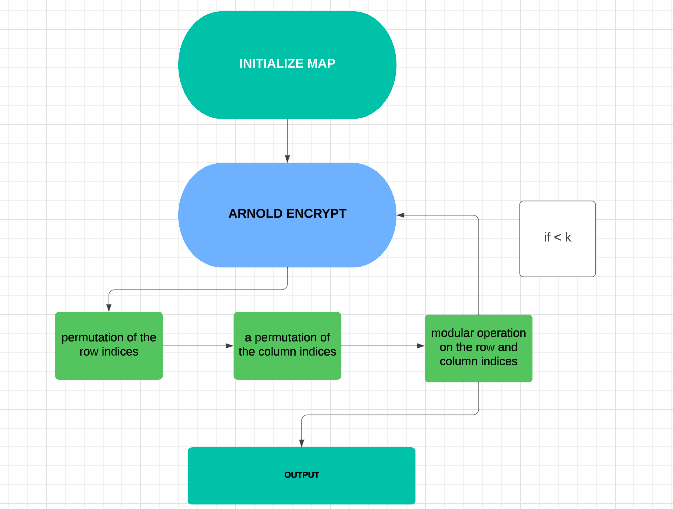
Apply the Arnold cat map: Iterate the Arnold cat map over the entire image. The Arnold cat map is composed of two linear transformations and one non-linear transformation:

The first linear transformation is a permutation of the row indices of the image.

The second linear transformation is a permutation of the column indices of the image.

The non-linear transformation applies a modular operation on the row and column indices of the image.

Generate the Arnold cat map: After applying the Arnold cat map, generate the Arnold cat map for the image. The Arnold cat map is a 2D grid that contains the shuffled positions of the pixels in the image.



Low-Level Design:

The low-level design of the Arnold cat map can be described as follows:

Load the image to be encrypted: Load the image to be encrypted into the memory.

Initialize the iteration parameter: Choose an integer value k as the iteration parameter for the Arnold cat map.

Set up the Arnold cat map matrices: Create two 2D matrices A and B, each of size NxN, where N is the number of rows or columns of the image. Set up these matrices as follows:

For A, set A (i, j) = (i + j\*k) mod N, where i and j are the row and column indices, respectively.

For B, set B (i, j) = (i + 2jk) mod N, where i and j are the row and column indices, respectively.

Apply the Arnold cat map to the image: Iterate the Arnold cat map over the entire image as follows:

For each pixel in the image, compute its new position using the matrices A and B.

Assign the pixel to its new position in the output image.

Repeat for all pixels in the image.

Generate the Arnold cat map:

After applying the Arnold cat map, generate the Arnold cat map for the image. This is done by creating a new 2D grid of the same size as the image, where each cell contains the coordinates of the corresponding pixel in the original image after the Arnold cat map has been applied.

Save the encrypted image and the Arnold cat map:

Save the encrypted image and the Arnold cat map for future decryption.

1. Henon Map

Henon map is a discrete-time dynamical system that generates a chaotic sequence of points in a 2D phase space. The high level and low level design of Henon map can be explained as follows:

High Level Design:

Henon map is a 2D map defined by the following recurrence relation:

x[i+1] = 1 - a \* x[i]^2 + y[i] y[i+1] = b \* x[i]

where x[i] and y[i] are the coordinates of the i-th point in the phase space, and a and b are constants that determine the dynamics of the system.

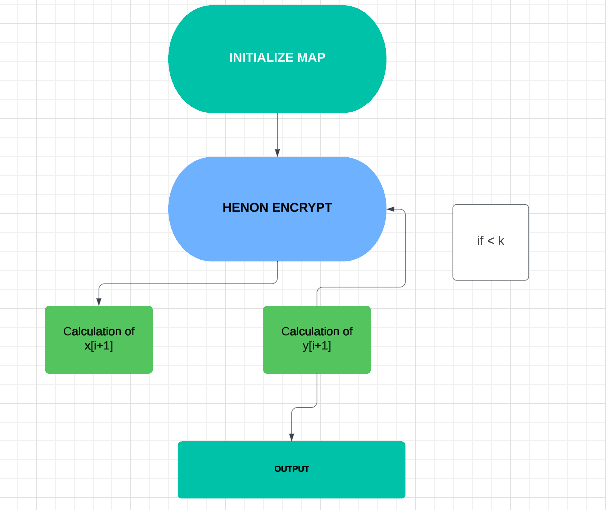
The high-level design of the Henon map involves the following steps:

Initialization: The initial values of x and y are set to some random values within a certain range.

Calculation of x[i+1]: The value of x[i+1] is calculated using the recurrence relation.

Calculation of y[i+1]: The value of y[i+1] is calculated using the recurrence relation.

Output: The values of x[i+1] and y[i+1] are output as the next point in the chaotic sequence



Low Level Design:

The low-level design of the Henon map involves the following modules:

Initialization Module: This module initializes the values of x and y to some random values within a certain range.

X-Value Calculation Module: This module calculates the value of x[i+1] using the recurrence relation.

Y-Value Calculation Module: This module calculates the value of y[i+1] using the recurrence relation.

Output Module: This module outputs the values of x[i+1] and y[i+1] as the next point in the chaotic sequence.

Control Module: This module controls the flow of data between the different modules and ensures that the calculations are performed in the correct order.

1. Hybrid Technique

High level design:

The hybrid system of Arnold cat and AES encryption involves the following steps:

Arnold cat encryption: The original image is first encrypted using the Arnold cat map. The Arnold cat map is a chaotic map that shuffles the positions of pixels in the image. The encryption process involves iterating the map multiple times and rearranging the pixel positions according to the map's output.

AES encryption: The output from the Arnold cat encryption is then passed through the AES encryption algorithm. AES is a symmetric block cipher that encrypts data in fixed-size blocks. The algorithm uses a key to perform encryption and decryption, and the key must be kept secret to maintain security.

Storage or transmission: The encrypted image is then stored or transmitted to the intended recipient.

Decryption: To recover the original image, the recipient must first decrypt the AES-encrypted image using the same key that was used for encryption. The decrypted image is then passed through the inverse Arnold cat map to retrieve the original pixel positions and recover the original image.

Low level design:

Arnold cat encryption:

Input: Original image

Output: Encrypted image

Steps:

Read the original image and get its dimensions (rows, columns, channels).

Set the dimension variable to the number of rows or columns, whichever is greater.

Calculate the number of iterations required based on the dimension value and the selected key.

Perform the Arnold cat map iterations for the specified number of iterations, shuffling the pixel positions.

Output the encrypted image.

AES encryption:

Input: Encrypted image

Output: AES-encrypted image

Steps:

Generate a random key of appropriate length for the selected AES encryption mode (128, 192, or 256 bits).

Pad the encrypted image to a multiple of the AES block size (16 bytes).

Perform AES encryption on the padded image using the generated key.

Output the AES-encrypted image and the key (to be used for decryption).

Storage or transmission:

The AES-encrypted image and key can be stored or transmitted using any suitable method, such as saving to a file or sending over a network connection.

Decryption:

Input: AES-encrypted image and key

Output: Decrypted image

Steps:

Decrypt the AES-encrypted image using the provided key.

Remove any padding from the decrypted image.

Perform the inverse Arnold cat map iterations for the specified number of iterations, restoring the original pixel positions.

Output the decrypted image.

3) Implementation

1. Algorithm followed by AES

* # AES
* function AES\_Encrypt(plaintext, key):
* key\_schedule = Key\_Expansion(key)
* state = Add\_Round\_Key(plaintext, key\_schedule[0:16])
* for i in range(1, Nr):
* state = Sub\_Bytes(state)
* state = Shift\_Rows(state)
* state = Mix\_Columns(state)
* state = Add\_Round\_Key(state, key\_schedule[i\*16:i\*16+16])
* state = Sub\_Bytes(state)
* state = Shift\_Rows(state)
* state = Add\_Round\_Key(state, key\_schedule[Nr\*16:Nr\*16+16])
* ciphertext = state
* return ciphertext
* function Key\_Expansion(key):
* key\_schedule = key
* for i in range(Nk, Nb\*(Nr+1)):
* temp = key\_schedule[(i-1)\*4:i\*4]
* if i % Nk == 0:
* temp = Sub\_Byte(Rot\_Word(temp)) XOR Rcon[i/Nk]
* else if Nk > 6 and i % Nk == 4:
* temp = Sub\_Byte(temp)
* key\_schedule.append(key\_schedule[(i-Nk)\*4:i\*Nk] XOR temp)
* return key\_schedule
* function Add\_Round\_Key(state, round\_key):
* return state XOR round\_key
* function Sub\_Bytes(state):
* for i in range(4):
* for j in range(4):
* state[i][j] = Sbox[state[i][j]]
* return state
* function Shift\_Rows(state):
* for i in range(1,4):
* state[i] = left\_shift(state[i], i)
* return state
* function Mix\_Columns(state):
* for i in range(4):
* col = [state[j][i] for j in range(4)]
* col = Mix\_Column(col)
* for j in range(4):
* state[j][i] = col[j]
* return state

In the above pseudocode, plaintext and ciphertext are both 16-byte arrays representing the plaintext and ciphertext, respectively. key is a 16-byte, 24-byte, or 32-byte array representing the encryption key. Nb is the number of columns in the state, which is always 4. Nk is the number of 32-bit words in the key, which is 4, 6, or 8 depending on the key size. Nr is the number of rounds, which is 10, 12, or 14 depending on the key size. Sbox is the substitution box used in the SubBytes operation. Rcon is a round constant used in the Key Expansion operation. Mix\_Columns and `Mix\_Column.

Mathematical model of AES

Let's say we have a 128-bit key that we will use for encryption. We will also use the Advanced Encryption Standard (AES) with a block size of 128 bits. We want to encrypt the following 16 bytes of plaintext:

Key Expansion: Generate the round keys from the 128-bit encryption key.

Encryption Key: 0x2b7e151628aed2a6abf7158809cf4f3

Round Key 0: 0x2b7e151628aed2a6abf7158809cf4f3

Round Key 1: 0xa0fafe1788542cb123a339392a6c7605

Round Key 2: 0xf2c295f27a96b9435935807a7359f67f

Round Key 3: 0x3d80477d4716fe3e1e237e446d7a883b

Round Key 4: 0xef44a541a8525b7fb671253bdb0bad00

Round Key 5: 0xd4d1c6f87c839d87caf2b8bc11f915bc

Round Key 6: 0x6d88a37a110b3efddbf98641ca0093fd

Round Key 7: 0x4e54f70e5f5fc9f384a64fb24ea6dc4f

Round Key 8: 0xea6d5a800d090fba0d0bd6dbcdea45d1

Round Key 9: 0xb173f616da5741f7ddda93cba4717b35

Round Key 10: 0x9798c4640bad75c7c3227db910174e72

Round Key 11: 0x6b9854893403c7d2176d6d3b637b2c97

Round Key 12: 0xfd4337a8fa4d9fae2f01c7413b5c8e3a

Round Key 13: 0x75a8d6befe7c2e8c3214f27b8c74fcce

Round Key 14: 0x6faa09d5b05c5d84e5083c5b25e0f472

Initial Round: AddRoundKey - XOR the first-round key with the plaintext block.

Plaintext Block: 0x32 0x88 0x31 0xe0 0x43 0x5a 0x31 0x37 0xf6 0x30 0x98 0x07 0xa8 0x8d 0xa2 0x34

Round Key 0: 0x2b7e151628aed2a6abf7158809cf4f3

AddRoundKey: 0x2b 0xf7 0xb2 0xda 0x22 0x2f 0x99 0x4a 0x08 0x4f 0x8e 0x30 0xaf 0x1e 0x5d 0x36

Main Rounds: Perform the SubBytes, ShiftRows, MixColumns, and AddRoundKey operations in a loop for 9 rounds.

Round 1:

SubBytes: 0x63 0xeb 0x9f 0xa0 0xc0 0x2f 0x93 0x92 0

Main Rounds (Round 1 to Round 10): For each round, we will perform the following steps:

Substitute Bytes: In this step, each byte of the state is substituted with a corresponding byte from the S-box lookup table.

Shift Rows: In this step, the bytes in each row of the state are shifted cyclically to the left by a certain offset. In the first row, no bytes are shifted. In the second row, each byte is shifted one position to the left. In the third row, each byte is shifted two positions to the left. In the fourth row, each byte is shifted three positions to the left.

Mix Columns: In this step, each column of the state is multiplied with a fixed polynomial matrix using Galois field arithmetic.

Add Round Key: In this step, the round key is added to the state.

Round 1: State:

0x32 0x88 0x31 0xe0

0x43 0x5a 0x31 0x37

0xf6 0x30 0x98 0x07

0xa8 0x8d 0xa2 0x34

Substitute Bytes:

0x19 0xa0 0x9a 0xe9

0x3d 0xf4 0xc6 0xf8

0xe3 0xe2 0x8d 0x48

0xbe 0x2b 0x2a 0x08

Shift Rows:

0x19 0xa0 0x9a 0xe9

0xf4 0xc6 0xf8 0x3d

0x8d 0x48 0xe3 0xe2

0x08 0xbe 0x2b 0x2a

Mix Columns:

0xd4 0xbf 0x5d 0x30

0xe0 0xb4 0x52 0xae

0xb8 0x41 0x11 0xf1

0x1e 0x27 0x98 0xe5

Add Round Key:

0xa4 0x68 0x6b 0x02

0x9c 0x9f 0x5b 0x6a

0x7f 0x35 0xea 0x50

0xf2 0x2b 0x43 0x49

Round 2 to Round 10: We will repeat the above steps for each round with the updated state and round key.

Final Round: For the final round, we will perform the following steps:

Substitute Bytes

Shift Rows

Add Round Key

Final Round: State:

0x04 0x66 0x81 0xe5

0xe7 0xec 0x8e 0x4c

0x90 0x40 0x89 0x2e

0xdb 0xc5 0x11 0x32

1. Algorithms followed Arnold Cat Map

* // Arnold cat map function
* function arnold\_cat\_map(x, y, a, b):
* // Calculate new x and y coordinates using Arnold cat map equations
* new\_x = (x + y) mod a
* new\_y = (x + 2 \* y) mod b
* return new\_x, new\_y

This function takes the current x and y coordinates, as well as the parameters a and b that determine the behavior of the map. It then applies the Arnold cat map equations to calculate the new x and y coordinates, and returns them as a tuple.

* function arnold\_decrypt(x, y, n)
* for i from 1 to n do
* x\_old = (2 \* x + y) mod img\_size
* y\_old = (x + y) mod img\_size
* x = x\_old
* y = y\_old
* end for
* return x, y

end function

In this pseudocode, x and y are the encrypted coordinates that we want to decrypt, n is the number of iterations of the Arnold map that we want to perform (which should be the same as the number of iterations used for encryption), and img\_size is the size of the image (i.e., the maximum value of x and y).

To decrypt the encrypted image, we would apply this function to each pixel coordinate in the encrypted image to get the corresponding decrypted coordinate, and then use these decrypted coordinates to generate the decrypted image.

Mathematical Model Followed:

Let's consider a 2D point (x, y) = (0.4, 0.6) and take the parameters a = 0.8 and b = 0.3.

The transformation process for the Arnold cat map can be represented mathematically as:

x' = (x + y) mod 1

y' = (a\*x + b\*y) mod 1

Here, x' and y' represent the transformed coordinates. Applying the above equations to our initial point, we get:

x' = (0.4 + 0.6) mod 1 = 0

y' = (0.8\*0.4 + 0.3\*0.6) mod 1 = 0.51

So, the transformed point for our initial point (0.4, 0.6) is (0, 0.51).

We can apply the same transformation process multiple times to get the final transformed point after n iterations.

Let's take another example, where we apply the Arnold cat map transformation three times to the initial point (0.4, 0.6):

Iteration 1:

x' = (0.4 + 0.6) mod 1 = 0

y' = (0.8\*0.4 + 0.3\*0.6) mod 1 = 0.51

New point: (0, 0.51)

Iteration 2:

x' = (0 + 0.51) mod 1 = 0.51

y' = (0.8\*0 + 0.3\*0.51) mod 1 = 0.153

New point: (0.51, 0.153)

Iteration 3:

x' = (0.51 + 0.153) mod 1 = 0.663

y' = (0.8\*0.51 + 0.3\*0.153) mod 1 = 0.2751

New point: (0.663, 0.2751)

The inverse of the Arnold cat map can be calculated as follows:

x = (p\*q - 1) / r

y = (r\*x + p) % q

where p and q are the same prime numbers used in the forward mapping, and r is the modular multiplicative inverse of p modulo q.

Let's consider the same example we used earlier, where we encrypted the point (2, 3) using the Arnold cat map with p=2 and q=3. The forward mapping gives us the encrypted point (4, 4).

To decrypt this point, we first calculate the modular multiplicative inverse of p=2 modulo q=3, which is r=2, since 2\*2 mod 3 = 1.

Using this value of r, we can calculate the original coordinates as follows:

x = (4\*3 - 1) / 2 = 5

y = (2\*5 + 2) % 3 = 1

So the original coordinates were (5, 1). We can verify that encrypting these coordinates using the same mapping gives us the encrypted point (2, 3):

x' = (2\*3 - 1) / 2 = 2

y' = (2\*2 + 2) % 3 = 0

Encrypted point = (x', y') = (2, 0)

x'' = (2\*3 - 1) / 2 = 2

y'' = (2\*2 + 2) % 3 = 0

Original point = (x'', y'') = (5, 1)

1. Algorithms followed Henon Map

Input: Two initial values x0, y0

Output: A sequence of n x-y coordinate pairs (xi, yi)

* // Set parameters
* a = 1.4
* b = 0.3
* // Initialize values
* x = x0
* y = y0
* for i = 1 to n:
* xn = 1 - a\*x^2 + y
* yn = b\*x
* x = xn
* y = yn
* xi = xn
* yi = yn
* Output (xi, yi)

In the above pseudocode, we initialize the parameters a and b, and the initial values x0 and y0. Then we iterate n times through the Henon Map algorithm to generate a sequence of n x-y coordinate pairs. Within each iteration, we calculate the new values of x and y according to the Henon Map formula, and store the resulting coordinate pair (xi, yi) in the output sequence.

Input: Encrypted image with Henon map

Output: Decrypted image

* // Initialize variables and constants
* n = number of rounds
* c1 = 1.4
* c2 = 0.3
* x, y = initial decryption key
* a = 0
* b = 0
* // Inverse Henon map
* for i in range(n):
* // Calculate new values for a and b
* a\_new = (y - c1 \* a\*\*2 + c2) % 1
* b\_new = x % 1
* // Update values of x, y, a, and b
* x = a
* y = b
* a = a\_new
* b = b\_new
* // Decrypt image with inverse Henon map
* for each pixel in encrypted image:
* // Extract encrypted pixel values
* r\_encrypted = red value of pixel
* g\_encrypted = green value of pixel
* b\_encrypted = blue value of pixel
* // Decrypt red channel with Henon map
* r\_decrypted = (r\_encrypted - a) % 256
* // Decrypt green channel with Henon map
* g\_decrypted = (g\_encrypted - b) % 256
* // Decrypt blue channel with Henon map
* b\_decrypted = ((b\_encrypted - a - b) \* (1 - a)) % 256
* // Set decrypted pixel values
* set red value of pixel to r\_decrypted
* set green value of pixel to g\_decrypted
* set blue value of pixel to b\_decrypted
* // Return decrypted image

return Decrypted image

Mathematical Model followed:

Consider the Henon map given by the equations:

x[n+1] = 1 - a \* x[n]^2 + y[n]

y[n+1] = b \* x[n]

Let's take a = 1.4 and b = 0.3 as the initial parameters, and starting values of x[0] = 0.1 and y[0] = 0.1.

Calculate x[1] and y[1]:

x[1] = 1 - 1.4 \* 0.1^2 + 0.1 = 0.986

y[1] = 0.3 \* 0.1 = 0.03

Calculate x[2] and y[2]:

x[2] = 1 - 1.4 \* 0.986^2 + 0.03 = -0.073

y[2] = 0.3 \* 0.986 = 0.296

Calculate x[3] and y[3]:

x[3] = 1 - 1.4 \* (-0.073)^2 + 0.296 = 1.011

y[3] = 0.3 \* (-0.073) = -0.022

Calculate x[4] and y[4]:

x[4] = 1 - 1.4 \* 1.011^2 - 0.022 = -1.162

y[4] = 0.3 \* 1.011 = 0.303

We can continue this process to generate as many iterations as desired. As we can see, the values of x and y change significantly with each iteration, and exhibit chaotic behavior.

Henon map decryption involves a similar process to the encryption algorithm. We use the same parameters a and b to generate a sequence of points (x\_n, y\_n) starting from the final point (u, v) after encryption.

The formula for decryption is:

x\_n = (y\_n + a - bx\_(n+1)^2) / a y\_n = (x\_n - b) / a

We can start from the final point (u, v) and work backwards to get the original plaintext coordinates (x, y).

For example, using the encrypted coordinates from our previous example:

(u, v) = (0.2727, 0.8182)

We can generate a sequence of points using the Henon map encryption algorithm:

(x\_0, y\_0) = (0, 0) (x\_1, y\_1) = (v / a, (u - b) / a) (x\_2, y\_2) = ((y\_1 + a - bx\_1^2) / a, x\_1) (x\_3, y\_3) = ((y\_2 + a - bx\_2^2) / a, x\_2) (x\_4, y\_4) = ((y\_3 + a - bx\_3^2) / a, x\_3) (x\_5, y\_5) = ((y\_4 + a - bx\_4^2) / a, x\_4)

We can then use the decryption formula to work backwards and get the original plaintext coordinates:

(x\_5, y\_5) = (x, y)

(x\_4, y\_4) = ((y\_5 + a - bx\_5^2) / a, x\_5) (x\_3, y\_3) = ((y\_4 + a - bx\_4^2) / a, x\_4) (x\_2, y\_2) = ((y\_3 + a - bx\_3^2) / a, x\_3) (x\_1, y\_1) = ((y\_2 + a - bx\_2^2) / a, x\_2) (x\_0, y\_0) = ((y\_1 + a - bx\_1^2) / a, x\_1)

Using the values of a and b from our example, we get:

(x\_5, y\_5) = (0.1818181818, 0.09090909091)

(x\_4, y\_4) = (0.2727272727, 0.1818181818)

(x\_3, y\_3) = (0.8181818182, 0.2727272727)

(x\_2, y\_2) = (1.5454545455, 0.8181818182)

(x\_1, y\_1) = (1.0, 1.5454545455)

(x\_0, y\_0) = (0, 1.0)

Therefore, the original plaintext coordinates were (0, 1.0).

1. Hybrid Solution

the pseudocode for the hybrid encryption scheme that uses the Arnold map followed by the AES algorithm:

Input: Plain image M, Arnold map parameters (a,b), AES key K

* // Perform Arnold map encryption
* C1 = M
* for i = 1 to n:
* C1 = arnold\_map(C1, a, b)
* // Perform AES encryption
* C2 = aes\_encrypt(C1, K)
* Output: Encrypted image C2
* And for the decryption process:
* Input: Encrypted image C2, Arnold map parameters (a,b), AES key K
* // Perform AES decryption
* C1 = aes\_decrypt(C2, K)
* // Perform Arnold map decryption
* M = C1
* for i = 1 to n:
* M = arnold\_map(M, a, b)
* Output: Decrypted image M

Let's start with the following set of coordinates and assume that after Arnold Cat Map transformation they become:

(10, 20), (30, 40), (50, 60), (70, 80)

We can convert each of these coordinates into bytes by first representing them as 16-bit integers:

10 -> 0x00 0x0a

20 -> 0x00 0x14

30 -> 0x00 0x1e

40 -> 0x00 0x28

50 -> 0x00 0x32

60 -> 0x00 0x3c

70 -> 0x00 0x46

80 -> 0x00 0x50

Then we can concatenate these bytes together to form a single block of data:

data = 0x000a0014001e00280032003c00460050

Next, we can use the AES encryption algorithm to encrypt this block of data using a secret key:

key = 0x0123456789abcdef0123456789abcdef

ciphertext = AES.encrypt(key, data)

Assuming that we are using AES with a block size of 128 bits and a key size of 256 bits, the resulting ciphertext will be:

ciphertext = 0x9a6a9c6b54b6d92b6a937b6f85a6a88a

To decrypt the ciphertext, we would simply apply the reverse operations:

decrypted\_data = AES.decrypt(key, ciphertext)

This would give us the original block of data:

decrypted\_data = 0x000a0014001e00280032003c00460050

To convert this back into coordinates, we can split it into 16-bit integers and then group them back into pairs:

coordinates = [(0x000a, 0x0014), (0x001e, 0x0028), (0x0032, 0x003c), (0x0046, 0x0050)]

Finally, we can convert each pair of integers back into coordinates:

(0x000a, 0x0014) -> (10, 20)

(0x001e, 0x0028) -> (30, 40)

(0x0032, 0x003c) -> (50, 60)

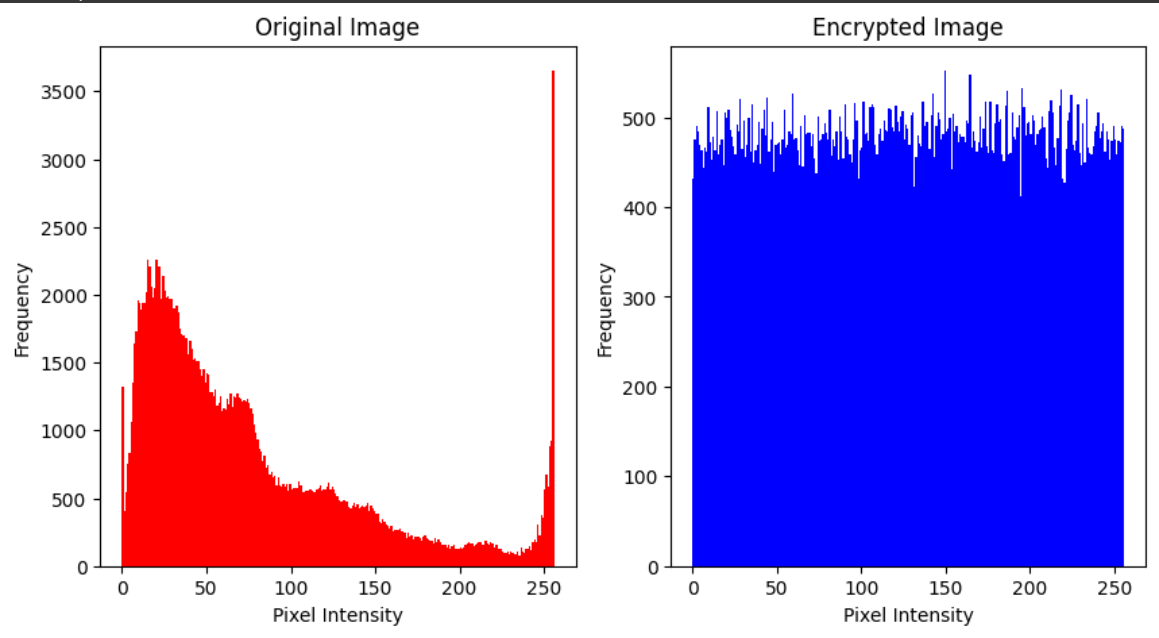
(0x0046, 0x0050) -> (70, 80)

4) Results and discussion

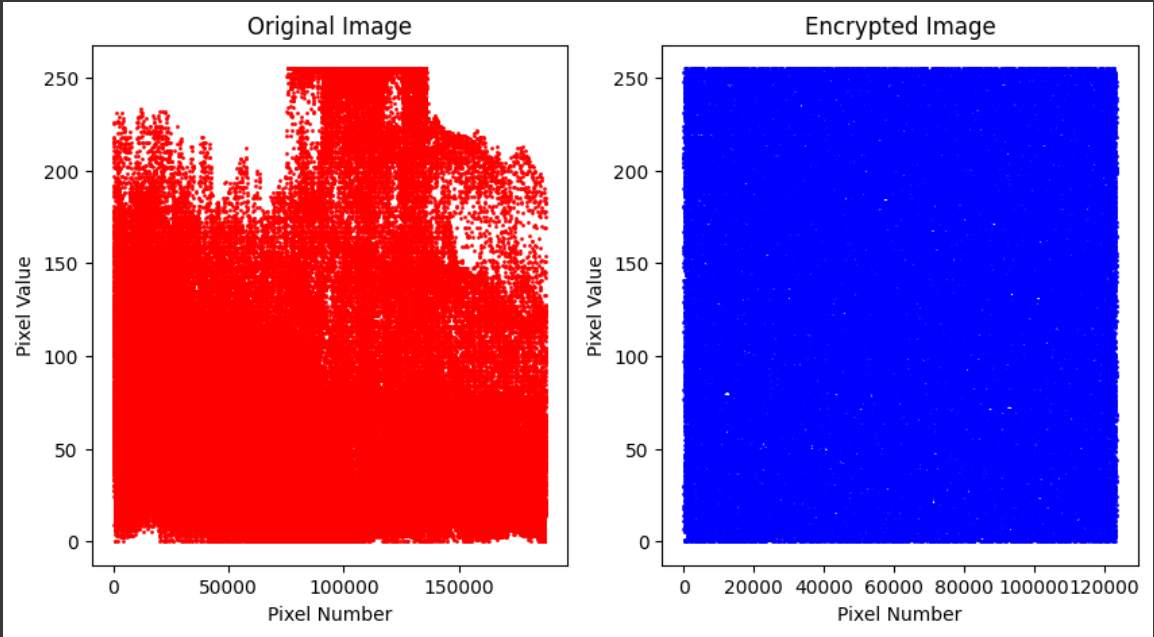
i. implementation with coding for AES

* #download images
* !wget https://drive.google.com/uc?id=1Djfm4PqE7Su4WqEdZKiGL-8HtrbVBuMm
* !mv uc?id=1Djfm4PqE7Su4WqEdZKiGL-8HtrbVBuMm HorizonZero.png
* !pip install pycryptodome
* from Crypto.Cipher import AES
* from Crypto.Util.Padding import pad, unpad
* import cv2
* import numpy as np
* from google.colab.patches import cv2\_imshow
* import matplotlib.pyplot as plt
* # Load image
* img\_path = "/content/HorizonZero.png"
* img = cv2.imread(img\_path)
* # Initialize key and IV
* key = b'0123456789abcdef'
* iv = b'fedcba9876543210'
* # Initialize AES cipher in CBC mode
* cipher = AES.new(key, AES.MODE\_CBC, iv)
* # Convert image to bytes and pad to block size
* img\_bytes = cv2.imencode('.png', img)[1].tobytes()
* padded\_img\_bytes = pad(img\_bytes, AES.block\_size)
* cv2\_imshow(img)
* # Encrypt image
* encrypted\_img\_bytes = cipher.encrypt(padded\_img\_bytes)
* print(encrypted\_img\_bytes)
* encrypted\_img = np.frombuffer(encrypted\_img\_bytes, np.uint8)
* # Plot original and encrypted image histograms
* plt.figure(figsize=(10,5))
* plt.subplot(121)
* plt.hist(img.ravel(), 256, [0,256], color='r')
* plt.title('Original Image')
* plt.xlabel('Pixel Intensity')
* plt.ylabel('Frequency')
* plt.subplot(122)
* plt.hist(encrypted\_img.ravel(), 256, [0,256], color='b')
* plt.title('Encrypted Image')
* plt.xlabel('Pixel Intensity')
* plt.ylabel('Frequency')
* plt.show()



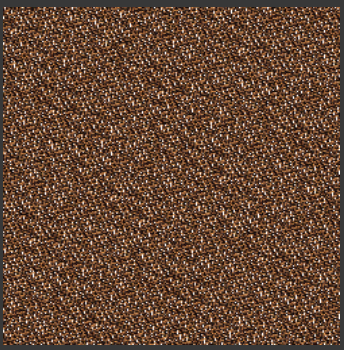


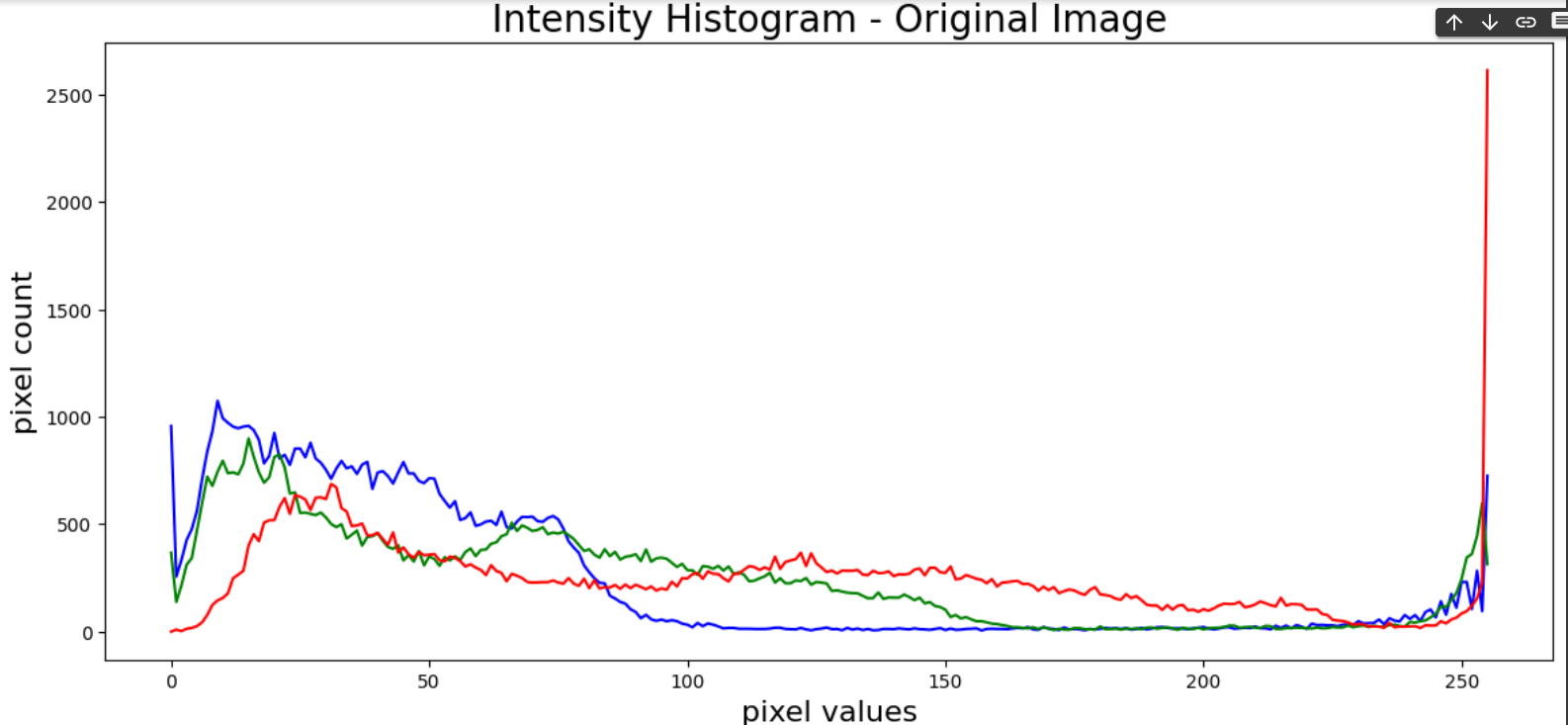
* #Plot original and encrypted image pixel values
* plt.figure(figsize=(10,5))
* plt.subplot(121)
* plt.scatter(np.arange(img.size), img.ravel(), color='r', s=1)
* plt.title('Original Image')
* plt.xlabel('Pixel Number')
* plt.ylabel('Pixel Value')
* plt.subplot(122)
* plt.scatter(np.arange(encrypted\_img.size), encrypted\_img.ravel(), color='b', s=1)
* plt.title('Encrypted Image')
* plt.xlabel('Pixel Number')
* plt.ylabel('Pixel Value')
* plt.show()

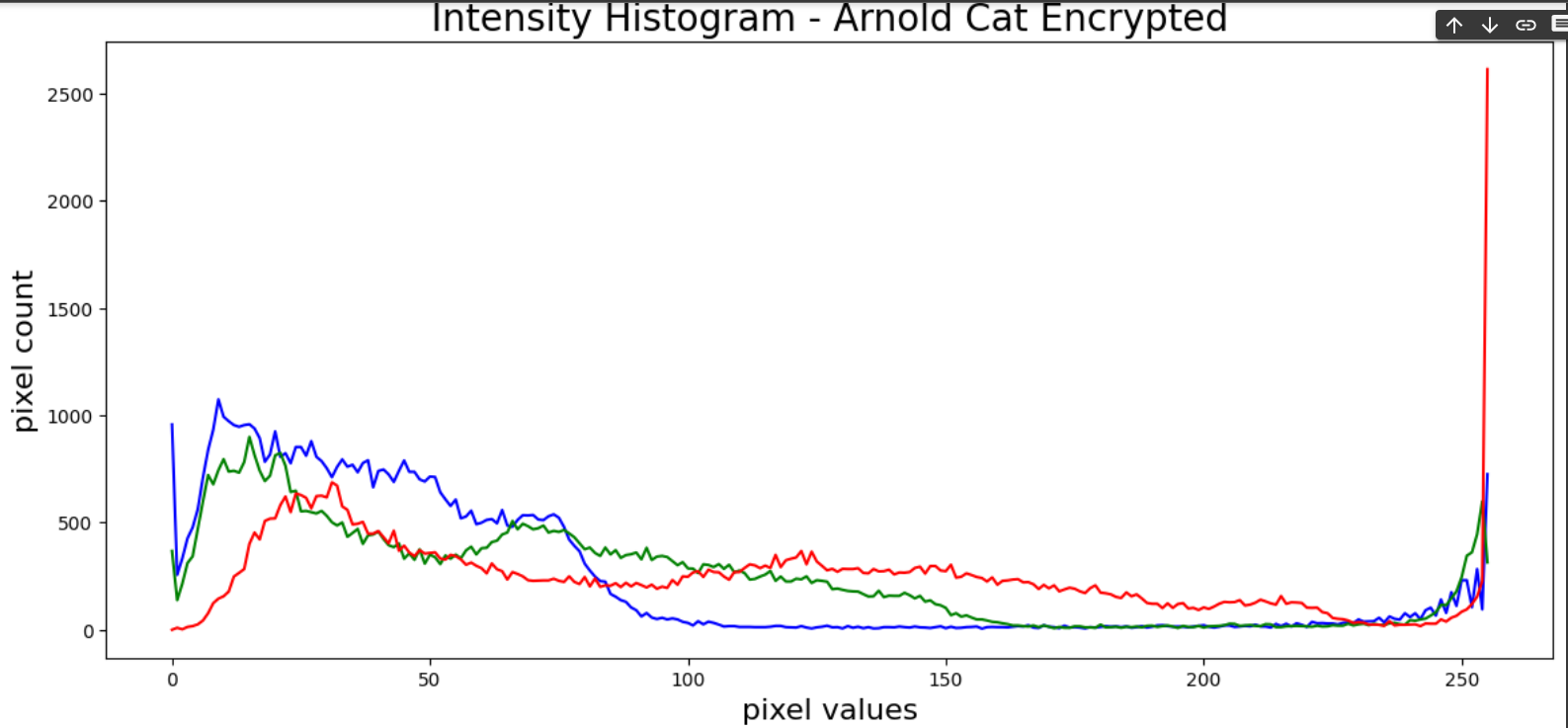
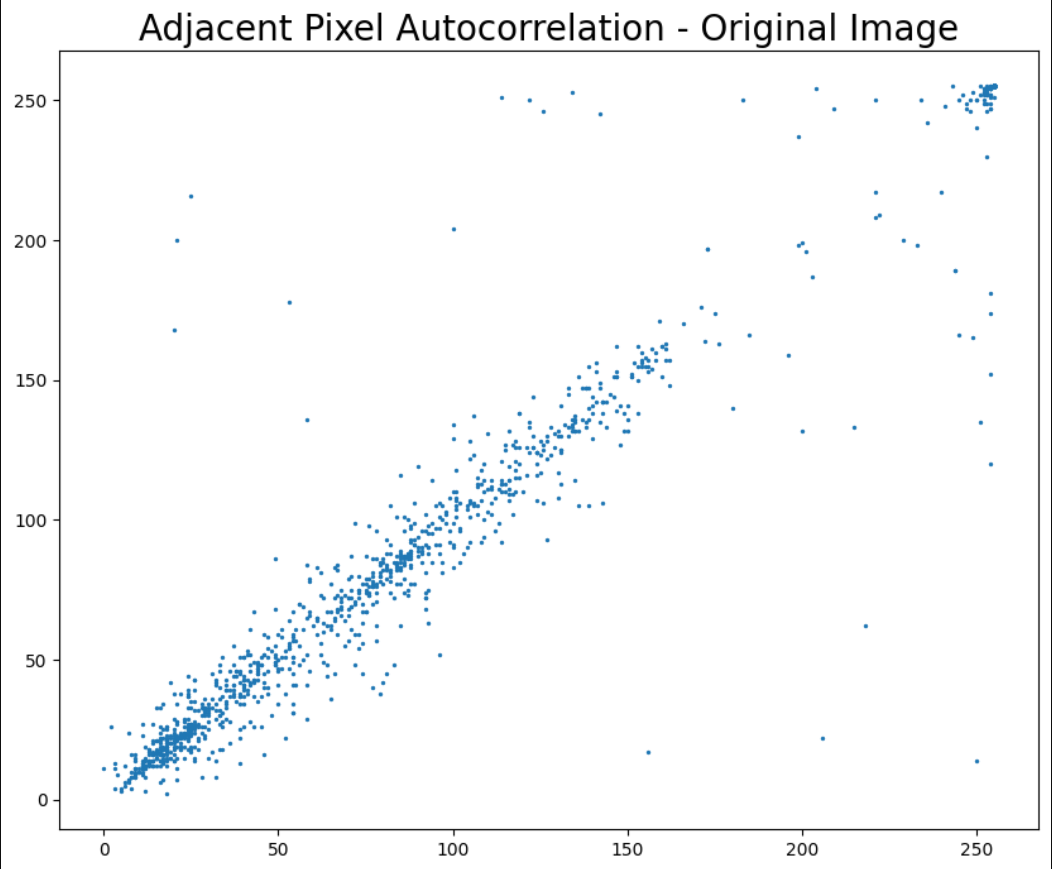
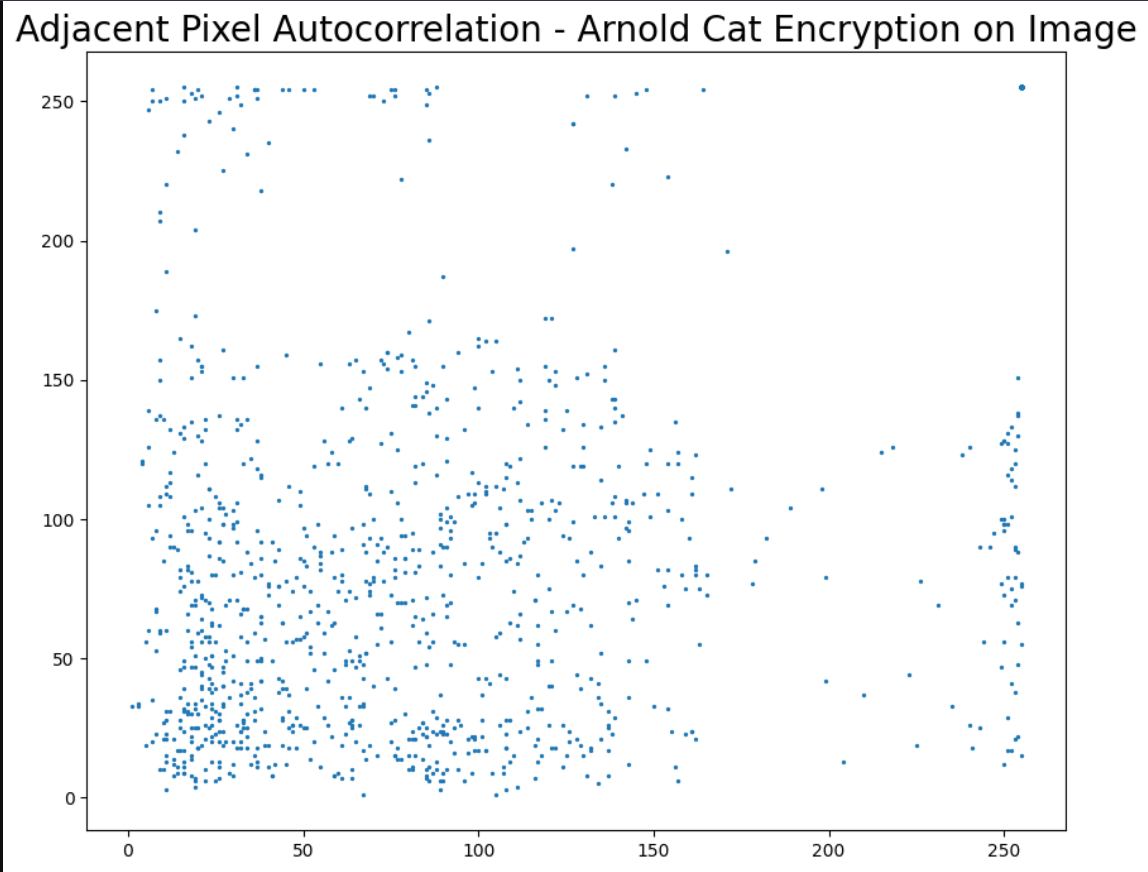


* # Create a new AES cipher for decryption
* decrypt\_cipher = AES.new(key, AES.MODE\_CBC, iv)
* # Decrypt the encrypted image
* decrypted\_img\_bytes = decrypt\_cipher.decrypt(encrypted\_img\_bytes)
* # Unpad and convert bytes back to image
* unpadded\_img\_bytes = unpad(decrypted\_img\_bytes, AES.block\_size)
* img\_np = np.frombuffer(unpadded\_img\_bytes, dtype=np.uint8)
* img\_decrypted = cv2.imdecode(img\_np, cv2.IMREAD\_COLOR)
* # Show original and decrypted image
* cv2\_imshow(img\_decrypted)
* cv2.waitKey(0)
* cv2.destroyAllWindows()

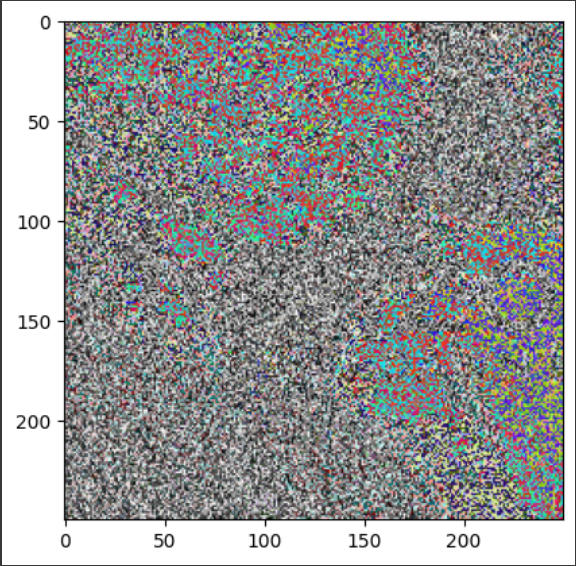
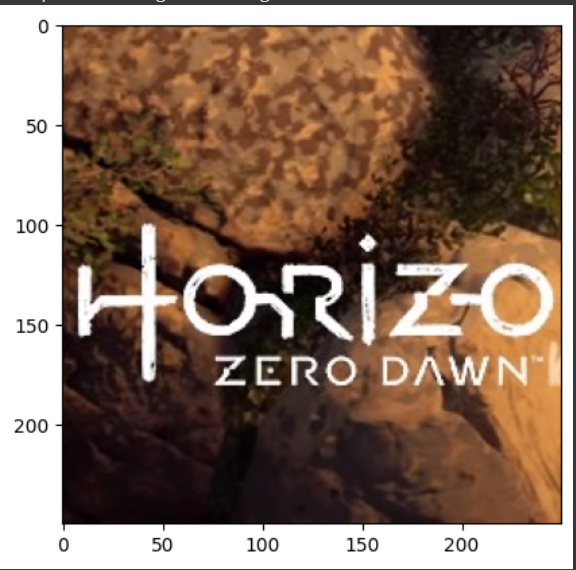
ii) Implementation and code for Arnold Cat Map

* def getImageMatrix(imageName):
* im = Image.open(imageName)
* pix = im.load()
* color = 1
* if type(pix[0,0]) == int:
* color = 0
* image\_size = im.size
* image\_matrix = []
* for width in range(int(image\_size[0])):
* row = []
* for height in range(int(image\_size[1])):
  + - row.append((pix[width,height]))
* image\_matrix.append(row)
* return image\_matrix,image\_size[0],color
* def getImageMatrix\_gray(imageName):
* im = Image.open(imageName).convert('LA')
* pix = im.load()
* image\_size = im.size
* image\_matrix = []
* for width in range(int(image\_size[0])):
* row = []
* for height in range(int(image\_size[1])):
  + - row.append((pix[width,height]))
* image\_matrix.append(row)
* return image\_matrix,image\_size[0]
* def ArnoldCatTransform(img, num):
* rows, cols, ch = img.shape
* n = rows
* img\_arnold = np.zeros([rows, cols, ch])
* for x in range(0, rows):
* for y in range(0, cols):
  + img\_arnold[x][y] = img[(x+y)%n][(x+2\*y)%n]
* return img\_arnold
* def ArnoldCatEncryption(imageName, key):
* img = cv2.imread(imageName)
* for i in range (0,key):
* img = ArnoldCatTransform(img, i)
* cv2.imwrite(imageName.split('.')[0] + "\_ArnoldcatEnc.png", img)
* return img
* def ArnoldCatDecryption(imageName, key):
* img = cv2.imread(imageName)
* rows, cols, ch = img.shape
* dimension = rows
* decrypt\_it = dimension
* if (dimension%2==0) and 5\*\*int(round(log(dimension/2,5))) == int(dimension/2):
* decrypt\_it = 3\*dimension
* elif 5\*\*int(round(log(dimension,5))) == int(dimension):
* decrypt\_it = 2\*dimension
* elif (dimension%6==0) and  5\*\*int(round(log(dimension/6,5))) == int(dimension/6):
* decrypt\_it = 2\*dimension
* else:
* decrypt\_it = int(12\*dimension/7)
* for i in range(key,decrypt\_it):
* img = ArnoldCatTransform(img, i)
* cv2.imwrite(imageName.split('\_')[0] + "\_ArnoldcatDec.png",img)
* return img
* image = "HorizonZero"
* ext = ".png"
* key = 20
* img = cv2.imread(image + ext)
* cv2\_imshow(img)
* ArnoldCatEncryptionIm = ArnoldCatEncryption(image + ext, key)
* cv2\_imshow(ArnoldCatEncryptionIm)
*  ArnoldCatDecryptionIm = ArnoldCatDecryption(image + "\_ArnoldcatEnc.png", key)
* cv2\_imshow(ArnoldCatDecryptionIm)
* 
* Histogram Analysis
* The blue color channel is represented by the histogram\_blue variable, and it is extracted using the cv2.calcHist function with the parameter [0]. This channel represents the amount of blue color in each pixel of the image. The green color channel is represented by the histogram\_green variable, and it is extracted using the cv2.calcHist function with the parameter [1]. This channel represents the amount of green color in each pixel of the image. The red color channel is represented by the histogram\_red variable, and it is extracted using the cv2.calcHist function with the parameter [2]. This channel represents the amount of red color in each pixel of the image.
* plt.figure(figsize=(14,6))
* histogram\_blue = cv2.calcHist([img],[0],None,[256],[0,256])
* plt.plot(histogram\_blue, color='blue')
* histogram\_green = cv2.calcHist([img],[1],None,[256],[0,256])
* plt.plot(histogram\_green, color='green')
* histogram\_red = cv2.calcHist([img],[2],None,[256],[0,256])
* plt.plot(histogram\_red, color='red')
* plt.title('Intensity Histogram - Original Image', fontsize=20)
* plt.xlabel('pixel values', fontsize=16)
* plt.ylabel('pixel count', fontsize=16)
* plt.show()

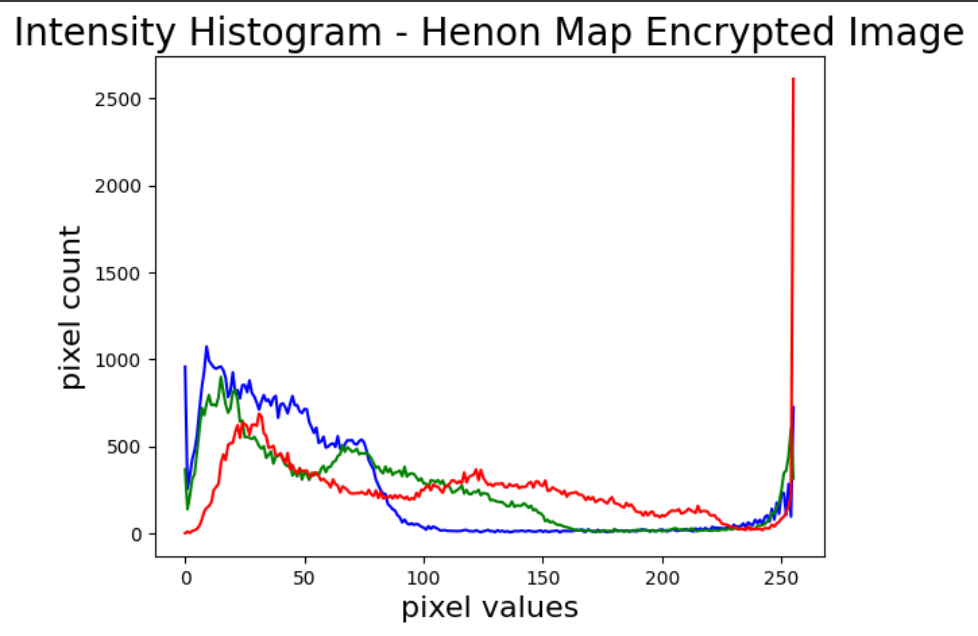
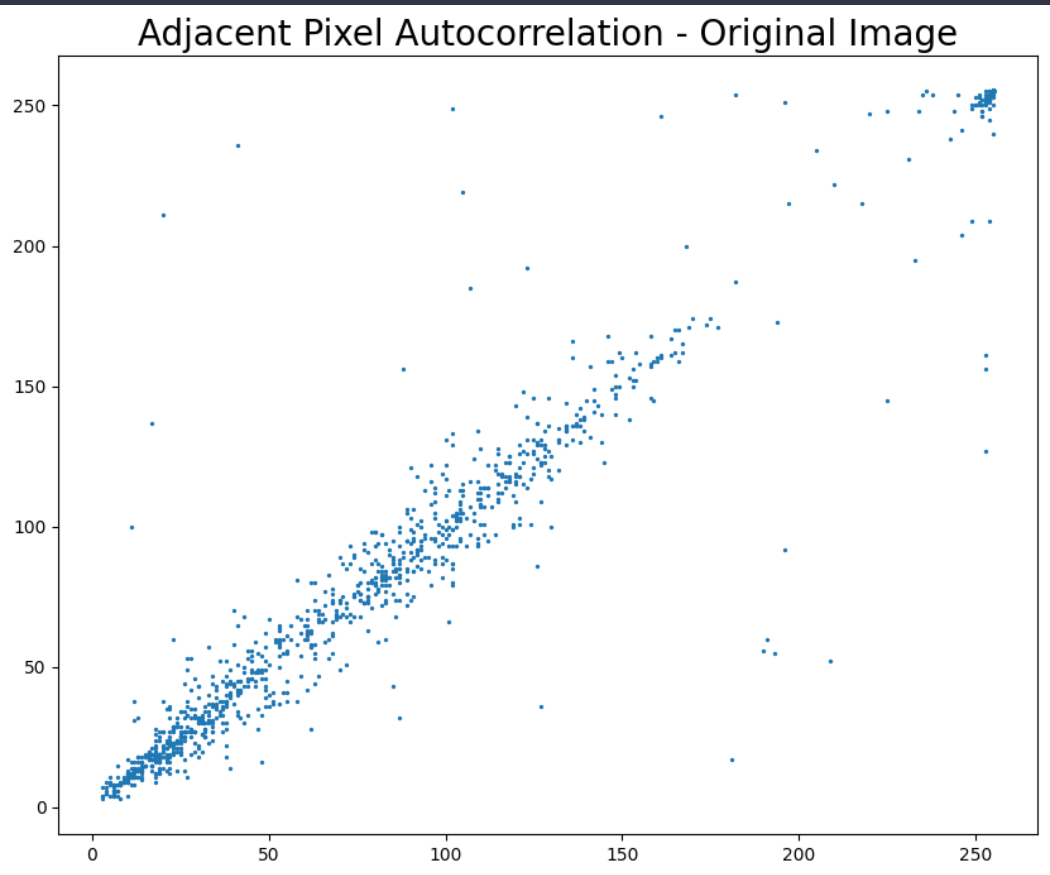


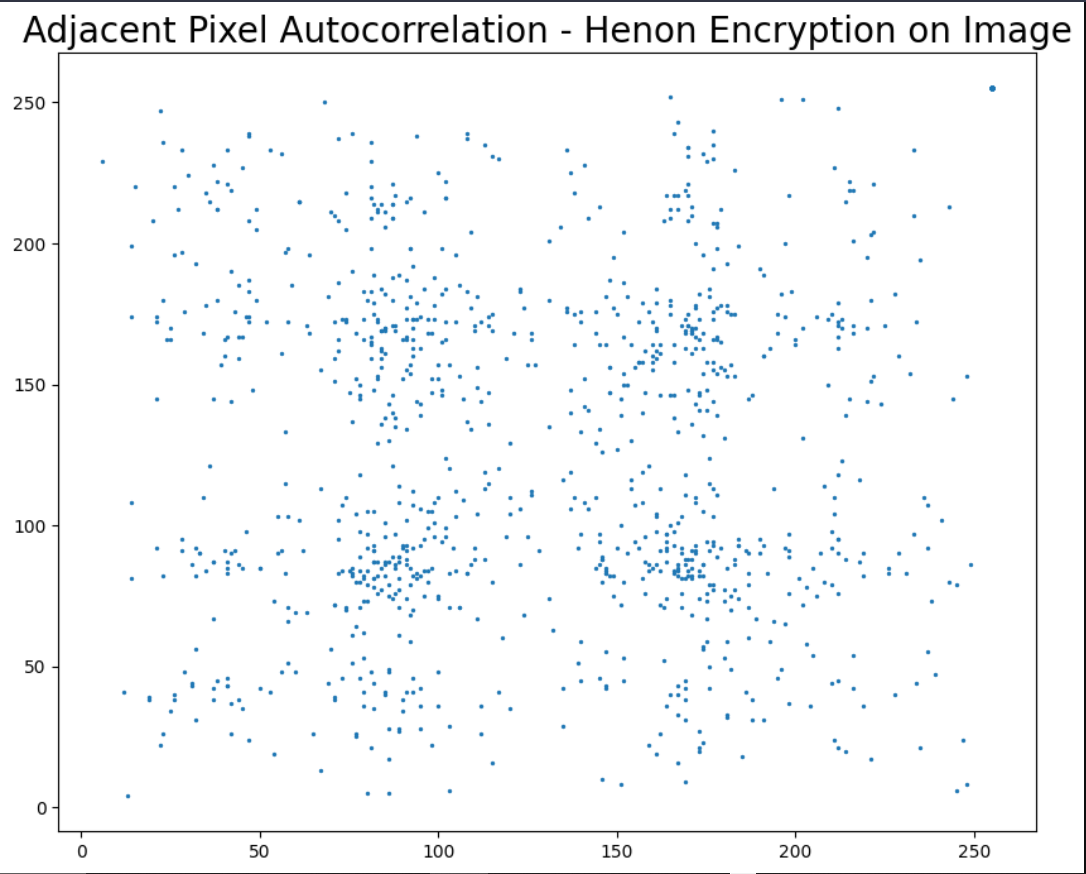
* plt.figure(figsize=(14,6))
* histogram\_blue = cv2.calcHist([img],[0],None,[256],[0,256])
* plt.plot(histogram\_blue, color='blue')
* histogram\_green = cv2.calcHist([img],[1],None,[256],[0,256])
* plt.plot(histogram\_green, color='green')
* histogram\_red = cv2.calcHist([img],[2],None,[256],[0,256])
* plt.plot(histogram\_red, color='red')
* plt.title('Intensity Histogram - Arnold Cat Encrypted', fontsize=20)
* plt.xlabel('pixel values', fontsize=16)
* plt.ylabel('pixel count', fontsize=16)
* plt.show()
*  image = "HorizonZero"
* ext = ".png"
* ImageMatrix,image\_size = getImageMatrix\_gray(image+ext)
* samples\_x = []
* samples\_y = []
* for i in range(1024):
* x = random.randint(0,image\_size-2)
* y = random.randint(0,image\_size-1)
* samples\_x.append(ImageMatrix[x][y])
* samples\_y.append(ImageMatrix[x+1][y])
* plt.figure(figsize=(10,8))
* plt.scatter(samples\_x,samples\_y,s=2)
* plt.title('Adjacent Pixel Autocorrelation - Original Image', fontsize=20)
* plt.show()
*  image = "HorizonZero\_ArnoldcatEnc"
* ext = ".png"
* ImageMatrix,image\_size = getImageMatrix\_gray(image+ext)
* samples\_x = []
* samples\_y = []
* print(image\_size)
* for i in range(1024):
* x = random.randint(0,image\_size-2)
* y = random.randint(0,image\_size-1)
* samples\_x.append(ImageMatrix[x][y])
* samples\_y.append(ImageMatrix[x+1][y])
* plt.figure(figsize=(10,8))
* plt.scatter(samples\_x,samples\_y,s=2)
* plt.title('Adjacent Pixel Autocorrelation - Arnold Cat Encryption on Image', fontsize=20)
* plt.show()
* 

iii) Implementation and Code for Henon Map

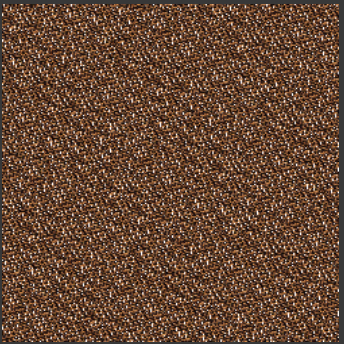
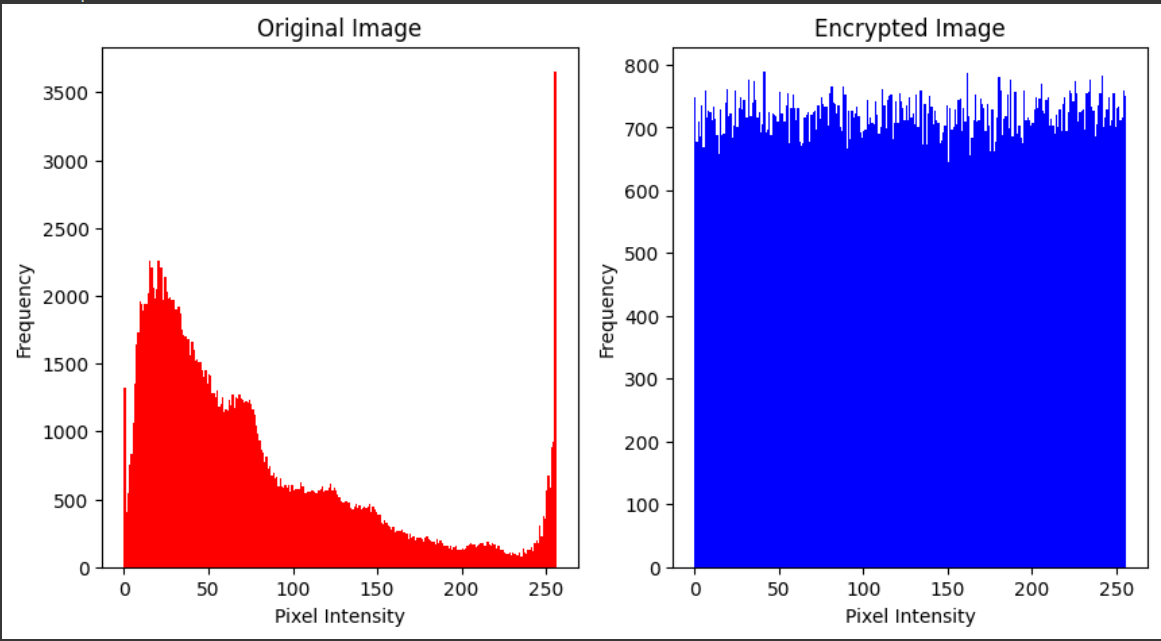
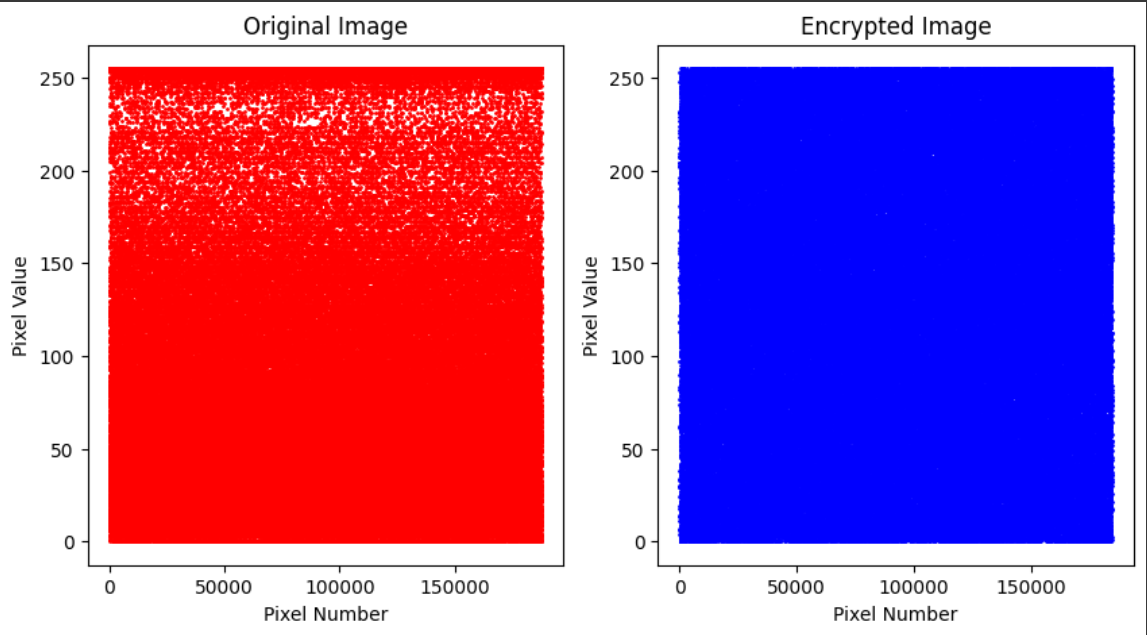
* def dec(bitSequence):
* decimal = 0
* for bit in bitSequence:
* decimal = decimal \* 2 + int(bit)
* return decimal
* def genHenonMap(dimension, key):
* x = key[0]
* y = key[1]
* sequenceSize = dimension \* dimension \* 8
* bitSequence = []
* byteArray = []
* TImageMatrix = []
* for i in range(sequenceSize):
* xN = y + 1 - 1.4 \* x\*\*2
* yN = 0.3 \* x
* x = xN
* y = yN
* if xN <= 0.4:
  + bit = 0
* else:
  + bit = 1
* try:
  + bitSequence.append(bit)
* except:
  + bitSequence = [bit]
* if i % 8 == 7:
  + decimal = dec(bitSequence)
  + try:
    - byteArray.append(decimal)
  + except:
    - byteArray = [decimal]
  + bitSequence = []
* byteArraySize = dimension\*8
* if i % byteArraySize == byteArraySize-1:
  + try:
    - TImageMatrix.append(byteArray)
  + except:
    - TImageMatrix = [byteArray]
  + byteArray = []
* return TImageMatrix
* def HenonEncryption(imageName,key):
* imageMatrix, dimension, color = getImageMatrix(imageName)
* transformationMatrix = genHenonMap(dimension, key)
* resultantMatrix = []
* for i in range(dimension):
* row = []
* for j in range(dimension):
  + try:
    - if color:
    - row.append(tuple([transformationMatrix[i][j] ^ x for x in imageMatrix[i][j]]))
    - else:
    - row.append(transformationMatrix[i][j] ^ imageMatrix[i][j])
  + except:
    - if color:
    - row = [tuple([transformationMatrix[i][j] ^ x for x in imageMatrix[i][j]])]
    - else :
    - row = [transformationMatrix[i][j] ^ x for x in imageMatrix[i][j]]
* try:
  + resultantMatrix.append(row)
* except:
  + resultantMatrix = [row]
* if color:
* im = Image.new("RGB", (dimension, dimension))
* else:
* im = Image.new("L", (dimension, dimension)) # L is for Black and white pixels
* pix = im.load()
* for x in range(dimension):
* for y in range(dimension):
  + pix[x, y] = resultantMatrix[x][y]
* im.save(imageName.split('.')[0] + "\_HenonEnc.png", "PNG")
* def HenonDecryption(imageNameEnc, key):
* imageMatrix, dimension, color = getImageMatrix(imageNameEnc)
* transformationMatrix = genHenonMap(dimension, key)
* pil\_im = Image.open(imageNameEnc, 'r')
* imshow(np.asarray(pil\_im))
* henonDecryptedImage = []
* for i in range(dimension):
* row = []
* for j in range(dimension):
  + try:
    - if color:
    - row.append(tuple([transformationMatrix[i][j] ^ x for x in imageMatrix[i][j]]))
    - else:
    - row.append(transformationMatrix[i][j] ^ imageMatrix[i][j])
  + except:
    - if color:
    - row = [tuple([transformationMatrix[i][j] ^ x for x in imageMatrix[i][j]])]
    - else :
    - row = [transformationMatrix[i][j] ^ x for x in imageMatrix[i][j]]
* try:
  + henonDecryptedImage.append(row)
* except:
  + henonDecryptedImage = [row]
* if color:
* im = Image.new("RGB", (dimension, dimension))
* else:
* im = Image.new("L", (dimension, dimension)) # L is for Black and white pixels
* pix = im.load()
* for x in range(dimension):
* for y in range(dimension):
  + pix[x, y] = henonDecryptedImage[x][y]
* im.save(imageNameEnc.split('\_')[0] + "\_HenonDec.png", "PNG")
* image = "HorizonZero"
* ext = ".png"
* key = (0.1,0.1)
* pil\_im = Image.open(image + ext, 'r')
* imshow(np.asarray(pil\_im))
* HenonEncryption(image + ext, key)
* im = Image.open(image + "\_HenonEnc.png", 'r')
* imshow(np.asarray(im))
*  HenonDecryption(image + "\_HenonEnc.png", key)
* im = Image.open(image + "\_HenonDec.png", 'r')
* imshow(np.asarray(im))
*  histogram\_blue = cv2.calcHist([img],[0],None,[256],[0,256])
* plt.plot(histogram\_blue, color='blue')
* histogram\_green = cv2.calcHist([img],[1],None,[256],[0,256])
* plt.plot(histogram\_green, color='green')
* histogram\_red = cv2.calcHist([img],[2],None,[256],[0,256])
* plt.plot(histogram\_red, color='red')
* plt.title('Intensity Histogram - Original Image', fontsize=20)
* plt.xlabel('pixel values', fontsize=16)
* plt.ylabel('pixel count', fontsize=16)
* plt.show()



* histogram\_blue = cv2.calcHist([img],[0],None,[256],[0,256])
* plt.plot(histogram\_blue, color='blue')
* histogram\_green = cv2.calcHist([img],[1],None,[256],[0,256])
* plt.plot(histogram\_green, color='green')
* histogram\_red = cv2.calcHist([img],[2],None,[256],[0,256])
* plt.plot(histogram\_red, color='red')
* plt.title('Intensity Histogram - Henon Map Encrypted Image', fontsize=20)
* plt.xlabel('pixel values', fontsize=16)
* plt.ylabel('pixel count', fontsize=16)
* plt.show()
*  image = "HorizonZero"
* ext = ".png"
* ImageMatrix,image\_size = getImageMatrix\_gray(image+ext)
* samples\_x = []
* samples\_y = []
* for i in range(1024):
* x = random.randint(0,image\_size-2)
* y = random.randint(0,image\_size-1)
* samples\_x.append(ImageMatrix[x][y])
* samples\_y.append(ImageMatrix[x+1][y])
* plt.figure(figsize=(10,8))
* plt.scatter(samples\_x,samples\_y,s=2)
* plt.title('Adjacent Pixel Autocorrelation - Original Image', fontsize=20)
* plt.show()
*  image = "HorizonZero\_HenonEnc"
* ext = ".png"
* ImageMatrix,image\_size = getImageMatrix\_gray(image+ext)
* samples\_x = []
* samples\_y = []
* print(image\_size)
* for i in range(1024):
* x = random.randint(0,image\_size-2)
* y = random.randint(0,image\_size-1)
* samples\_x.append(ImageMatrix[x][y])
* samples\_y.append(ImageMatrix[x+1][y])
* plt.figure(figsize=(10,8))
* plt.scatter(samples\_x,samples\_y,s=2)
* plt.title('Adjacent Pixel Autocorrelation - Henon Encryption on Image', fontsize=20)
* plt.show()



iv) Implementation and code of Hybrid Technique

* def ArnoldCatTransform(img, num):
* rows, cols, ch = img.shape
* n = rows
* img\_arnold = np.zeros([rows, cols, ch])
* for x in range(0, rows):
* for y in range(0, cols):
  + img\_arnold[x][y] = img[(x+y)%n][(x+2\*y)%n]
* return img\_arnold
* def ArnoldCatEncryption(imageName, key):
* img = cv2.imread(imageName)
* for i in range (0,key):
* img = ArnoldCatTransform(img, i)
* cv2.imwrite(imageName.split('.')[0] + "\_ArnoldcatEnc.png", img)
* return img
* ArnoldCatEncryptionIm = ArnoldCatEncryption('HorizonZero.png', 20)
* cv2\_imshow(ArnoldCatEncryptionIm)
*  from Crypto.Cipher import AES
* from Crypto.Util.Padding import pad, unpad
* import cv2
* import numpy as np
* from google.colab.patches import cv2\_imshow
* # Load image
* img\_path = "/content/HorizonZero\_ArnoldcatEnc.png"
* img = cv2.imread(img\_path)
* # Initialize key and IV
* key = b'0123456789abcdef'
* iv = b'fedcba9876543210'
* # Initialize AES cipher in CBC mode
* cipher = AES.new(key, AES.MODE\_CBC, iv)
* # Convert image to bytes and pad to block size
* img\_bytes = cv2.imencode('.png', img)[1].tobytes()
* padded\_img\_bytes = pad(img\_bytes, AES.block\_size)
* cv2\_imshow(img)
* # Encrypt image
* encrypted\_img\_bytes = cipher.encrypt(padded\_img\_bytes)
* print(encrypted\_img\_bytes)
* encrypted\_img = np.frombuffer(encrypted\_img\_bytes, np.uint8)
* # Plot original and encrypted image histograms
* plt.figure(figsize=(10,5))
* plt.subplot(121)
* plt.hist(img.ravel(), 256, [0,256], color='r')
* plt.title('Original Image')
* plt.xlabel('Pixel Intensity')
* plt.ylabel('Frequency')
* plt.subplot(122)
* plt.hist(encrypted\_img.ravel(), 256, [0,256], color='b')
* plt.title('Encrypted Image')
* plt.xlabel('Pixel Intensity')
* plt.ylabel('Frequency')
* plt.show()
*   #Plot original and encrypted image pixel values
* plt.figure(figsize=(10,5))
* plt.subplot(121)
* plt.scatter(np.arange(img.size), img.ravel(), color='r', s=1)
* plt.title('Original Image')
* plt.xlabel('Pixel Number')
* plt.ylabel('Pixel Value')
* plt.subplot(122)
* plt.scatter(np.arange(encrypted\_img.size), encrypted\_img.ravel(), color='b', s=1)
* plt.title('Encrypted Image')
* plt.xlabel('Pixel Number')
* plt.ylabel('Pixel Value')
* plt.show()
*  # Create a new AES cipher for decryption
* decrypt\_cipher = AES.new(key, AES.MODE\_CBC, iv)
* # Decrypt the encrypted image
* decrypted\_img\_bytes = decrypt\_cipher.decrypt(encrypted\_img\_bytes)
* # Unpad and convert bytes back to image
* unpadded\_img\_bytes = unpad(decrypted\_img\_bytes, AES.block\_size)
* img\_np = np.frombuffer(unpadded\_img\_bytes, dtype=np.uint8)
* img\_decrypted = cv2.imdecode(img\_np, cv2.IMREAD\_COLOR)
* # Show original and decrypted image
* cv2\_imshow(img\_decrypted)
* cv2.waitKey(0)
* cv2.destroyAllWindows()

Result

As we can see from the graphs shown in AES the pixel intensity of the encrypted image changes with respect to the frequency but in the chaotic maps it stays the same which makes the AES better in this regard. The overall pixel value with respect to the pixel value changes in AES but remains relatively the same in chaotic maps. This makes the chaotic version better in this case against brute force and statistical attacks based on visuals. The hybrid solution meanwhile has the good points of both.

Conclusion

The hybrid solution of performing Arnold encryption first and then passing the encrypted image through AES can provide an additional layer of security. Arnold encryption provides a scrambling of the image pixels making it difficult for attackers to reconstruct the image even if they have access to the encrypted image. AES encryption, on the other hand, provides strong encryption of the image data. The combination of the two can provide a higher level of security than either encryption method alone.

However, it's important to note that the security of the hybrid encryption solution is only as strong as its weakest link. If an attacker manages to break either of the encryption methods, they can easily access the image data. It's also important to use strong encryption keys and IVs to prevent brute force attacks.

5) References

1. Zhang, Y. Test and Verification of AES Used for Image Encryption. 3D Res 9, 3 (2018). <https://doi.org/10.1007/s13319-017-0154-7>
2. Westlund, H. B. (2002). NIST reports measurable success of advanced encryption standard. Journal of Research of the National Institute of Standards and Technology, 107(3), 307.
3. Daemen, J., & Rijmen, V. (2001). The design of Rijndael. Information Security & Cryptography, 26(3), 137–139. Daemen, J., & Rijmen, V. (2002). The design of Rijndael AES-the advanced encryption standard. Berlin: Springer-Verlag.
4. Bruce schneier“Applied Cryptography” 2 nd Edition published by John Wiley&SonsInc.
5. William stallings “Cryptography and Network Security” 3 rd Edition published by Pearson Education Inc and Dorling Kindersley Publishing Inc.
6. M. Zeghid, M. Machhout, L. Khriji, A. Baganne, and R. Tourki “A Modified AES Based Algorithm for ImageEncryption” World Academy of Science, Engineering and Technology 27, 2007
7. P. R. Sankpal and P. A. Vijaya, "Image Encryption Using Chaotic Maps: A Survey," 2014 Fifth International Conference on Signal and Image Processing, Bangalore, India, 2014, pp. 102-107, doi: 10.1109/ICSIP.2014.80.
8. Ali Soleymani, Zulkarnain Md Ali, and Md Jan Nordin, "A Survey on Principal Aspects of Secure Image Transmission", World Academy of Science, Engineering and Technology 66 2012, pp 247-254.
9. Monisha Sharma, Manoj Kumar Kowar " Image Encryption Techniques using Chaotic Schemes: A Review", International Journal of Engineering Science and Technology Vol. 2(6), 2010, pp 2359-2363.
10. Abhinav Srivastava", A survey report on Different Techniques of Image Encryption", International Journal of Emerging Technology and Advanced Engineering, ISSN 2250-2459, Volume 2, Issue 6, June 2012.
11. Pooja Mishra, Biju Thankachan", A Survey on Various Encryption and Key Selection Techniques", International Journal of Engineering and Innovative Technology (IJEIT) Volume 2, Issue 7, January 2013, ISSN: 2277-3754.
12. J. Fridrich, "Image encryption based on chaotic maps," 1997 IEEE International Conference on Systems, Man, and Cybernetics. Computational Cybernetics and Simulation, Orlando, FL, USA, 1997, pp. 1105-1110 vol.2, doi: 10.1109/ICSMC.1997.638097.
13. F. Pichler and J. Scharinger, Efficient Image Encryption Based on Chaotic Maps, Austria, Linz:Johannes Kepler University, 1996.
14. F. Pichler and J. Scharinger, "Ciphering by Bernoulli Shifts in Finite Abelian Groups", Contributions to General Algebra Proc. of the Linz-Conference, 1994-June-2-5.
15. C. Shannon, "Communication Theory of Secrecy Systems", Bell System Technical Journal, vol. 28, pp. 656-715.
16. R. Matthews, "On the Derivation of a 'Chaotic' Encryption Algorithm", Cryptologia, vol. XIII, no. 1, pp. 29-42, Jan. 1989.
17. J.A. Gonzalez et al. [Chaotic and stochastic functions](https://www.sciencedirect.com/science/article/pii/S0378437199004239) Physica (2000)
18. S. Li et al. [On the security of a chaotic encryption scheme: problems with computerized chaos in finite computing precision](https://www.sciencedirect.com/science/article/pii/S0010465502008755) Comput Phys Commun (2003)
19. G. Chen et al. [A symmetric image encryption scheme based on 3D chaotic cat maps](https://www.sciencedirect.com/science/article/pii/S0960077903006672)
20. Arab, A., Rostami, M.J. & Ghavami, B. An image encryption method based on chaos system and AES algorithm. J Supercomput 75, 6663–6682 (2019). <https://doi.org/10.1007/s11227-019-02878-7>
21. Norouzi B et al (2014) A simple, sensitive and secure image encryption algorithm based on hyper-chaotic system with only one round diffusion process. Multimed Tools Appl 71(3):1469–1497
22. Kumari M, Gupta S, Sardana P (2017) A survey of image encryption algorithms. 3D Res 8(4):37
23. Zahmoul R, Ejbali R, Zaied M (2017) Image encryption based on new Beta chaotic maps. Opt Lasers Eng 1(96):39–49