IMAGE ENCRYPTION AND DECRYPTION SCHEME USING LOGISTIC CHAOTIC MAP AND PIXEL SUBSTITUTION

A PROJECT REPORT

Submitted by

JEYASRI MEENAKSHI K 2019103530

NIRUTHIYA SHRI P 2019103549

RAMYA DEVI P 2019103568

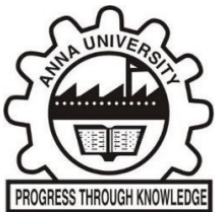
in partial fulfilment of the award of the degree

of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING



COLLEGE OF ENGINEERING, GUINDY CAMPUS

ANNA UNIVERSITY : CHENNAI 600 025

MAY 2023

BONAFIDE CERTIFICATE

Certified that this project report titled **IMAGE ENCRYPTION AND DECRYPTION SCHEME USING LOGISTIC CHAOTIC MAP AND PIXEL SUBSTITUTION** is the *bonafide* work of **JEYASRI MEENAKSHI K (2019103530), NIRUTHIYA SHRI P (2019103549)** and **RAMYA DEVI P (2019103568)** who carried out the project work under my supervision,for the fulfilment of the requirements for the award of the degree of Bachelor of Engineering in Computer Science and Engineering. Certified further that to the best of my knowledge, the work reported herein does not form part of any other thesis or dissertation on the basis of which a degree or an award was conferred on an earlier occasion on these or any other candidates.

Place : Chennai
Date: 29.5.23

M.S.Karthika Devi
Assistant Professor
Department of Computer Science and Engineering
Anna University, Chennai - 25

COUNTERSIGNED

Head of the Department,

Department of Computer Science and Engineering,

Anna University Chennai,

Chennai - 600025

ACKNOWLEDGEMENT

We express our deep gratitude to our guide, **M.S.Karthika Devi**, Assistant Professor, Department of Computer Science and Engineering for guiding us through every phase of the project. We appreciate her thoroughness, tolerance and ability to share her knowledge with us. We would also like to thank her for her kind support and for providing necessary facilities to carry out the work.

We express our thanks to the panel of reviewers **Dr.V.Mary Anita Rajam**, Professor, Department of Computer Science and Engineering and **Dr. S. Sudha**, Professor Department of Computer Science and Engineering for their valuable suggestions and critical reviews throughout the course of our project.

We are extremely grateful to **Dr. S. Valli**, Professor & Head of the Department, Department of Computer Science and Engineering, Anna University, Chennai – 25, for extending the facilities of the Department towards our project and for her unstinting support.

We express our thanks to all other teaching and non-teaching staff who helped us in one way or other for the successful completion of the project. We would also like to thank our parents, family and friends for their indirect contribution in the successful completion of this project.

JEYASRI MEENAKSHI K **NIRUTHIYA SHRI P** **RAMYA DEVI P**

ABSTRACT - ENGLISH

With the widespread use of digital communication and storage, there is an increasing need to protect sensitive images from unauthorised access or tampering.By encrypting images, any unauthorised modification or tampering attempts will render the image unreadable or invalid, providing a mechanism to detect and prevent unauthorised alterations.The traditional encryption schemes provides lesser levels of security even though they involve more computational complexity.To overcome this problem, we have proposed a image encryption scheme that has multiple levels of image encryption thus enhancing the security of the image.In addition the overall work also includes decrypting the encrypted image provided that the user has a correct key in hand.The key obtained through Secure Hash Algorithm(SHA) is encrypted further using Elliptic curve cryptography.The key streams generated from logistic chaotic map are then used to permute the pixels of the image to provide properties of confusion and diffusion which provides adequate security against differential-like attacks and fast encryption.Besides, the discrete cosine transform(DCT) is used to transform the images into the frequency domain. With the help of a substitution box sufficient substitution of pixels is achieved.The resulting image is then encrypted using RSA encryption for additional security during transmission to the receiver.On the receiving end, the above mentioned processes are carried out in reverse order to retrieve the original image.

ABSTRACT - TAMIL

தற்போது டிஜிட்டல் தகவல்தொடர்பு மற்றும் சேமிப்பகத்தில் படங்கள் பரவலாக பயன்படுத்தப்படுகிறது. அதனால் அங்கீகரிக்கப்படாத அணுகல் அல்லது சேதத்திலிருந்து முக்கியமான படங்களை பாதுகாக்க வேண்டிய அவசியம் அதிகரித்து வருகிறது.படங்களை குறியாக்குவதன் மூலம், அங்கீகரிக்கப்படாத மாற்றம் அல்லது சேதப்படுத்தும் முயற்சிகளால் படத்தை படிக்க முடியாத அல்லது செல்லாததாக மாற்றும் ஒரு சூழலை இது உருவாக்குகிறது. இதனால் அங்கீகரிக்கப்படாத மாற்றங்களைக கண்டறிந்து தடுப்பதற்கான ஒரு பொறிமுறையை குறியாக்கம் வழங்குகிறது.பாரம்பரிய குறியாக்கத் திட்டங்கள் அதிக கணக்கீட்டு சிக்கலை உள்ளடக்கியிருந்தாலும் குறைந்த அளவிலான பாதுகாப்பையே வழங்குகிறது.இந்த சிக்கலை சமாளிக்க, பட குறியாக்கத்தின் பல நிலைகளைக் கொண்ட ஒரு பட குறியாக்கத் திட்டத்தை நாங்கள் முன்மொழிந்தோம். இதனால் படங்களின் பாதுகாப்பு அதிக அளவில் மேம்படுத்துப்படுகிறது. கூடுதலாக இந்த படைப்பில்,சரியான விசையைக் கொண்டு குறியாக்கப்பட்ட படத்தை மறைகுறியாக்கி அசல் படத்தை பெற முடியும். பாதுகாப்பான ஹாஷ் அல்காரிதம் (SHA) மூலம் பெறப்பட்ட

விசை, எலிப்டிக் வளைவு குறியாக்கவியலைப் பயன்படுத்தி மேலும் குறியாக்கம் செய்யப்பட்டுகிறது. லாஜிஸ்டிக் குழப்பமான வரைபடத்திலிருந்து கிடைக்கப்பட்ட விசை வரிசகள் படத்தின் படத்துணுக்கை வரிசைப்படுத்தப் பயன்படுத்தப்படுகின்றன.இதனால் படத்துணுக்கு குழப்பம் மற்றும் பரவல் செய்யப்பட்டுவதால், படமானது வேகமாக குறியாக்கப்படுவதுடன் வேறுபட்ட தாக்குதல்களுக்கு எதிராக போதுமான பாதுகாப்பையும் பெறுகிறது.மேலும், தனித்துவமான கோசைன் உருமாற்றம் (DCT) மூலம் படங்கள் அதிர்வெண் களத்திற்குள் மாற்றப்படுகிறது.மாற்று பெட்டியின் உதவியுடன் படத்துணுக்கு போதுமான சீரற்ற மாற்றீடை அடைகிறது.இதன் விளைவாக வரும் படம் ஆரெஸே(RSA) குறியாக்கத்தைப் பயன்படுத்தி குறியாக்கம் செய்யப்படுகிறது.இதனால் படத்தை பெறும் நபர்க்கு பரிமாற்றம் செய்யும் போது படம் கூடுதல் பாதுகாப்பை பெறுகிறது.பெறும் முடிவில், மேலே குறிப்பிடப்பட்ட செயல்முறைகள் அசல் படத்தை மீட்டெடுப்பதற்காக தலைகீழ் வரிசையில் மேற்கொள்ளப்படுகின்றன.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS, ABBREVIATIONS AND NOMENCLATURE

AES	Advanced Encryption Standard
CC	Correlation Coefficient
CML	Coupled Map Lattices
CNCM	Coupled Nonlinear Chaotic Map
cov(p)	Covariance of the pixels of the images
CS	Compressive Sensing
DCC	Diagonal Correlation Coefficient
DCT	Discrete Cosine Transform
DES	Data Encryption Standard
D(p)	Variance of the pixels of the image
ECC	Elliptic Curve Cryptography
GCD	Greatest Common Divisor
H	Height of the image
HCC	Horizontal Correlation Coefficient
IDCT	Inverse Discrete Cosine Transform
IDEA	International Data Encryption Algorithm
LSIC	Latin Square Image Cipher
M(p)	Mean of pixels of image
NPCR	Number of Pixels Change Rate
p	Pixel of image
RC4	Rivest Cipher 4
RGB	Red,Green and Blue
RSA	Rivest Shamir Adleman
S	Total number of pixels in image
SHA	Secure Hash Algorithm
SPN	Substitution Permutation Network
UACI	Unified Average Changing Intensity

VCC	Vertical Correlation Coefficient
W	Width of the image
XOR	Exclusively-OR

CHAPTER 1
INTRODUCTION

With the rapid development of multimedia technology, digital image has become an increasingly prevalent information carrier over the Internet. With the awakening of awareness that data is the treasure, personal privacy information contained in digital images is attracting more and more attention from malicious attackers. Image security is of great importance in both military and commercial fields.The illegal digital image handling makes it necessary to protect images.Unexpected exposure of private photos and divulged military and governmental classified images emphasises the importance of image security again and again.With the fast development of digital storage devices, computers, and the world-wide network, a digital image can be easily copied to mobile storage or transferred to the other side of the world within a second. However, such convenience could also be used by malicious/ unauthorised users to rapidly spread the image information such that it may cause uncountable losses for the owner(s) of images.

Image multimedia contains a lot of information, and the image is open in the process of communication. The security of image information will be threatened in the process of transmission, especially in the low security channel. Therefore, improving the security of image information in the process of storage and transmission, and avoiding the leakage of image multimedia information is an urgent problem to be solved.

A large number of image-oriented privacy-preserving techniques are constantly emerging, for instance, image encryption,image digital signature,secret image sharing, and image information hiding.Image encryption is the most straightforward way.Encryption helps hiding information to make it impenetrable without special knowledge.Image Encryption is the process in which the original image is encoded in such a way that only authorised parties

can read it. It doesn't prevent image interception but denies the image content to the interceptor. This distorted image can be decrypted by the person having knowledge about the decryption key. However, it is still possible to decrypt the distorted image without possessing the key. This is known as cryptanalysis. Significant attempts are being made to come up with algorithms which can make cryptanalysis difficult.Two basic techniques to obscure high redundancies and strong correlations are confusion (substitution) and diffusion (permutation). Confusion increases the complexity between the key and the cipher text and bans all attempts to study the cipher text for redundancies and statistical patterns. Diffusion, on the other hand, spreads the redundancy of the plaintext over the entire cipher text, thus decreasing redundancy. While either of these techniques alone is highly susceptible to being cracked, they generally make an excellent security solution when combined. Images, however, have various intrinsic features, such as bulk data capacity and high correlation among pixels, that render traditional encryption algorithms such as DES, IDEA unsuitable.

1.1 OBJECTIVES

- To explore how logistic chaotic maps can be used for ciphering and deciphering images.
- To implement an overall scheme for encryption and decryption of human facial images using various techniques such as SHA hashing, logistic chaotic map and substitution permutation network(SPN), Discrete Cosine Transformation and RSA.
- To analyse and bring out how the proposed scheme is much more efficient in terms of security and computational power than the state-of-the-art algorithms such as AES.
- To ensure enhanced security of key using Secure hashing Algorithm for initial value generation.

- To carry out frequency domain based image encryption using discrete cosine transform technique(DCT).
- To create an encryption and decryption scheme in levels that is much more efficient in terms of security and computational power in order to achieve security against differential attacks and fast encryption.
- To regenerate the decrypted image as similar to the original image as possible without loss of information.

1.2 BACKGROUND STUDY

Chaos has become one of the most important research fields in the past decade. Almost all the scholars are trying to apply it in the secure communication and cryptography field with chaotic control or synchronisation. However, the traditional encryption algorithm is less efficient when applied to the system that encrypts a large number of pictures or encrypts video. The chaotic system has superior properties in the field of information encryption.Chaos-based algorithms have shown some exceptionally good properties in many concerned aspects regarding security, complexity, speed, computing power, computational overhead and has sensitivity to initial value conditions, unpredictability and bifurcation complexity.

Chaos has some complex properties that contribute to the generation of more secure and robust encryption algorithms. Chaotic models include Arnold Cat map, Logistic map, Lorenz, Henon map, and some other models. One-dimensional chaotic system with the advantages of high-level efficiency and simplicity such as Logistic map has been widely used now.This complex nature of chaotic mapping can be reflected in the characteristics of certain encryption processes similar to ideal ciphers, such as diffusion, aliasing, balancing, and avalanche.Therefore, recent researches of image encryption algorithms have been increasingly based on chaotic systems, though the

drawbacks of small key space and weak security in one-dimensional chaotic cryptosystems are obvious.

Chaos theory has been established since the 1970s by many different research areas, such as physics, mathematics, engineering, biology, etc. Since the 1990s, many researchers have noticed that there exists a close relationship between chaos and cryptography.Many properties of chaotic systems have their corresponding counterparts in traditional cryptosystems. Chaotic systems have several significant features favourable to secure communications, such as ergodicity, sensitivity to initial conditions, control parameters and random-like behaviour, which can be connected with some conventional cryptographic properties of good ciphers, such as confusion and diffusion.It can be found from the definition and characteristics of the chaotic system that the chaotic system is very sensitive to the initial value. In a cryptographic system, if the subtle changes in the key can lead to obvious changes in the encryption results, the encryption algorithm or the cryptographic system has a better encryption effect, that is, the high sensitivity to the existence of the key. Therefore, with the sensitivity of the chaotic system to the initial value, an encryption system based on chaotic systems would be the wiser option.

1.3 PROBLEM STATEMENT

The primary problem to be addressed is to provide a much more secure encryption scheme which would be fast with less computational overhead and low complexity than traditional approaches such as AES,DES etc.So here we have explored how these limitations could be addressed using a novel encryption scheme.The initial value generation based on SHA algorithm is much more secure as vulnerabilities has not been found till now and it's never been broken. Enhancing the security by SPN has been studied and tried. In addition how DCT further improves the encryption quality and how it

transforms to frequency domain is also studied. Further how the proposed scheme is much more efficient in terms of security and computational power than state-of-the-art algorithms such as AES has been identified. Besides how logistic chaotic maps stand out from other kinds of chaotic maps has been compared through various evaluation metrics.

1.4 ORGANISATION OF THE THESIS

The thesis is organised as follows. Chapter 2 discusses the existing methods in the literature for image encryption. Chapter 3 introduces the architecture diagram along with the details of each of the modules. It elaborates on each of the component modules along with the obtained output. Chapter 4 discusses the performance metric used in the work and the values we obtained. A comparative analysis of the results is also done there. Chapter 5 concludes the present work while discussing the criticisms and scope for future work.

CHAPTER 2
LITERATURE SURVEY

The method of designing an image oriented CS-based product cipher with the help of chaos has been discussed in the work done by Rui Zhang et.al[20] . Considering that a CS-based cipher is a lightweight cipher, they give a supplementary way to improve the security. Finally, a novel image sampling–compression–encryption framework is proposed for secure image acquisition, in which a high-level confidentiality is achieved through two approaches. One is that both the measurement matrix and sparsifying basis controlled by a same chaotic system are regarded as the shared secrets between the encoder and the decoder. Specifically, the elements of the measurement matrix come from a chaotic sequence controlled by a tent map, and the column vectors of a selected sparsifying basis are randomly permuted by leveraging the same tent map. Even having figured out either measurement matrix or on a sparsifying basis, attackers are still unable to reconstruct the original signal from CS measurements successfully. The other is that the captured data are further encrypted to break the linearity constraint of CSbased cipher by performing the sample-level permutation and the bit-level XOR, which are also manipulated by a tent map. Obviously, diffusion and confusion effects can be easily achieved due to good pseudo randomness. In essence, the proposed imaging system is a two-layer cryptosystem consisting of CS-based cipher and chaos based encryption.

An image encryption scheme using complete shuffling, transformation of substitution box, and predicated image crypto-system has been proposed[8]. This proposed algorithm presents extra confusion in the first phase because of including an S-box based on using substitution by AES algorithm in encryption and its inverse in Decryption. In the second phase, shifting and rotation were used based on a secret key in each channel depending on the result from the

chaotic map, 2D logistic map and the output was processed and used for the encryption algorithm. It is known from earlier studies that simple encryption of images based on the scheme of shuffling is insecure in the face of chosen ciphertext attacks. Later, an extended algorithm has been projected. This algorithm performs well against chosen ciphertext attacks. In addition, the proposed approach was analysed for NPCR, UACI (Unified Average Changing Intensity), and Entropy analysis for determining its strength.

Chaotic map based permutation substitution and Latin square image cipher encryption scheme for images has been discussed in the work of H.T. Panduranga et.al[14].The proposed method consists of a permutation and substitution process. In the permutation process, a plain image is permuted according to a chaotic sequence generated using a chaotic map. In the substitution process, based on a secret key of 256 bit generate a Latin Square Image Cipher (LSIC) and this LSIC is used as key image and perform XOR operation between permuted image and key image. The proposed method can be applied to any plain image with unequal width and height as well and also resist statistical attack, differential attack. Experiments carried out for different images of different sizes. The proposed method possesses a large key space to resist brute force attack.

Another significant work using a spatiotemporal chaotic system for colour images using R,G,B matrix has been proposed in the work of Xue-Mei Bao[18].Initially, they use the R, G and B components of a colour plain-image to form a matrix. Then the matrix is permuted by using zigzag path scrambling. The resultant matrix is then passed through a substitution process. Finally, the ciphered colour image is obtained from the confused matrix. Theoretical analysis and experimental results indicate that the proposed scheme is both secure and practical, which make it suitable for encrypting colour images of any size.To overcome the problem of neglecting the correlations between R,G and B components,a novel colour image encryption algorithm based on chaos has been

proposed in this work. We use CML to encrypt the colour image and make the three components affect each other. The permutation and substitution stages effectively reduce the correlations between R, G and B components and enhance the encryption performance.

Anirban Bhowmick et.al[6] this work brings forth a simple but secure symmetric key encryption algorithm. The image encryption approach makes use of two pseudorandom number generators to encrypt images. Middle Square Algorithm is used to swap the columns of the image followed by the swapping of the rows. This permutation step produces the intermediary cipher image. Subsequently, RC4 algorithm is used to produce a stream of pseudo-random numbers. These numbers are used to substitute the intensity of pixels of the intermediary cipher image, which gives the final encrypted image. Various analysis tests are performed on the quality of the encrypted image.

A high-dimension Lorenz chaotic system and perceptron model within a neural network, a chaotic image encryption system with a perceptron model is a novel approach in the image encryption domain[17]. A low-dimensional chaotic system to design one encryption algorithm is very easy, but some defects of the low-dimensional chaotic system, like the simple dynamic characteristic, simple cipher code, and less determined sequence parameters, may cause serious security problems. Compared with the low-dimension ones, the high-dimensional chaotic system has a more complex structure, more system variables, and parameters. All of these characters assure it can be used as a cipher key of cryptosystem. Then the cryptosystem's key space will be larger, and the system variables' time sequence will be more erratic and unpredictable than using the low-dimensional chaotic system. So it is better to choose a high-dimensional chaotic system to design the cryptosystem. This paper chooses the Lorenz system as the encrypted text. The Lorenz system has a complex inner structure. When the system parameters vary, the system will show different period's chaotic dynamic behaviour.

Tao Li et.al[10] states that, a good encryption algorithm needs the following characteristics: after encryption, the grey histogram of ciphertext image is more average; Good entropy of information; Low correlation of ciphertext; Can resist differential attack; High sensitivity to secret keys; The secret key space is large enough; Anti-noise attack and so on. proposes a new image encryption algorithm based on two-dimensional Lorenz and Logistic.The classical chaotic model is used in the encryption algorithm to generate two sets of chaotic sequences to encrypt the image. The two-dimensional Lorenz chaotic model is used to generate chaotic sequences to encrypt and encrypt the image. Through the security analysis it can be concluded that the image encryption algorithm proposed in their work is sensitive to the secret key and has a large secret key space. It can resist exhaustive attacks to a high degree and can resist noise interference. The algorithm has strong security and robustness and is suitable for image encryption with high security level.The image encryption scheme can be used to encrypt grayscale images, as well as colour images. The colour image encryption step is: separating the R component, G component and B component of the colour image, encrypting the three components separately using the proposed scheme for grey image encryption, and finally combining the ciphertext image into a colour ciphertext image.

A Coupled Nonlinear Chaotic Map, called CNCM, and a novel chaos-based image encryption algorithm to encrypt colour images by using CNCM is discussed in the work done by Sahar Mazloom et.al[12]. The chaotic cryptography technique is a symmetric key cryptography with a stream cipher structure. In order to increase the security of the proposed algorithm, a 240 bit-long secret key is used to generate the initial conditions and parameters of the chaotic map by making some algebraic transformations to the key. These transformations as well as the nonlinearity and coupling structure of the CNCM have enhanced the cryptosystem security. For getting higher security and higher complexity, the current work employs the image size and colour components to

cryptosystem,thereby significantly increasing the resistance to known/chosen-plaintext attacks.Firstly, the image (P) of size N × M is converted into its RGB components. Afterwards, each color's matrix (R, G or B) is converted into a vector of integers within {0, 1, … , 255}. Each vector has a length of L = N × M. Then, the obtained three vectors represent the plaintext P(3 × L) which will be encrypted.

A double logistic chaotic map based image encryption has been put forward in the work of Hailan Pan[15].In the double cha-otic digital image encryption, second-level logistic chaotic map is mainly used to create and generate pseudo random sequence numbers, and the number of random sequences of image confusion and scrambling is obtained through two creation processes. In the process of encryption and decryption, the key used in the double chaotic digital image encryption method is the calculation parameter of the first-level logistic chaotic map and the initial value of the second-level logistic chaotic map. The encryption process is performed in the order of confusion and scrambling, and the decryption process is processed in the reverseorder.The simulation results of the pictures show that the histogram, pixel correlation, information entropy, key space size, and key sensitivity all reach a high level, and the image decryption processing can be completed basically correctly.

CHAPTER 3
SYSTEM DESIGN

3.1.ARCHITECTURE DIAGRAM

A hash value using SHA 512 algorithm from the original image is computed which is text wrapped and 4 parts of key are generated.The keys are encrypted using elliptic curve cryptography public key.Using logistic chaotic map key streams are generated from non encrypted values.Then the rows and columns of the image array are permuted and the permuted image array is converted back to an image.The encrypted image from row permutation is taken as the input. The colour channels of the image are separated. Then DCT is applied to each colour channel and then the colour channels are stacked to get the DCT encrypted image.A substitution based on the parameters of the image is generated. Then a random key which is generated as the same size of the substitution box is used for substituting pixels.In RSA public key and private key are generated.Using those keys the image is encrypted as the final output of encryption. In decryption,first for RSA the private key is generated. Using that RSA decrypted image is obtained.In substitution,the key is decrypted. Using the substitution box and the decrypted key the pixels are substituted to get the pixel substituted image.Then colour channels are separated. Then for each channel IDCT is applied and the colour channels are stacked to get the decrypted image.The keys are decrypted using elliptic curve cryptography private key. Using these decrypted values as the input ,the logistic chaotic map generates the keystream.Using that keystream first column permutation and then row permutation is performed to get the final decrypted image.

The overall architecture diagram for the proposed scheme is shown in Figure 3.1.

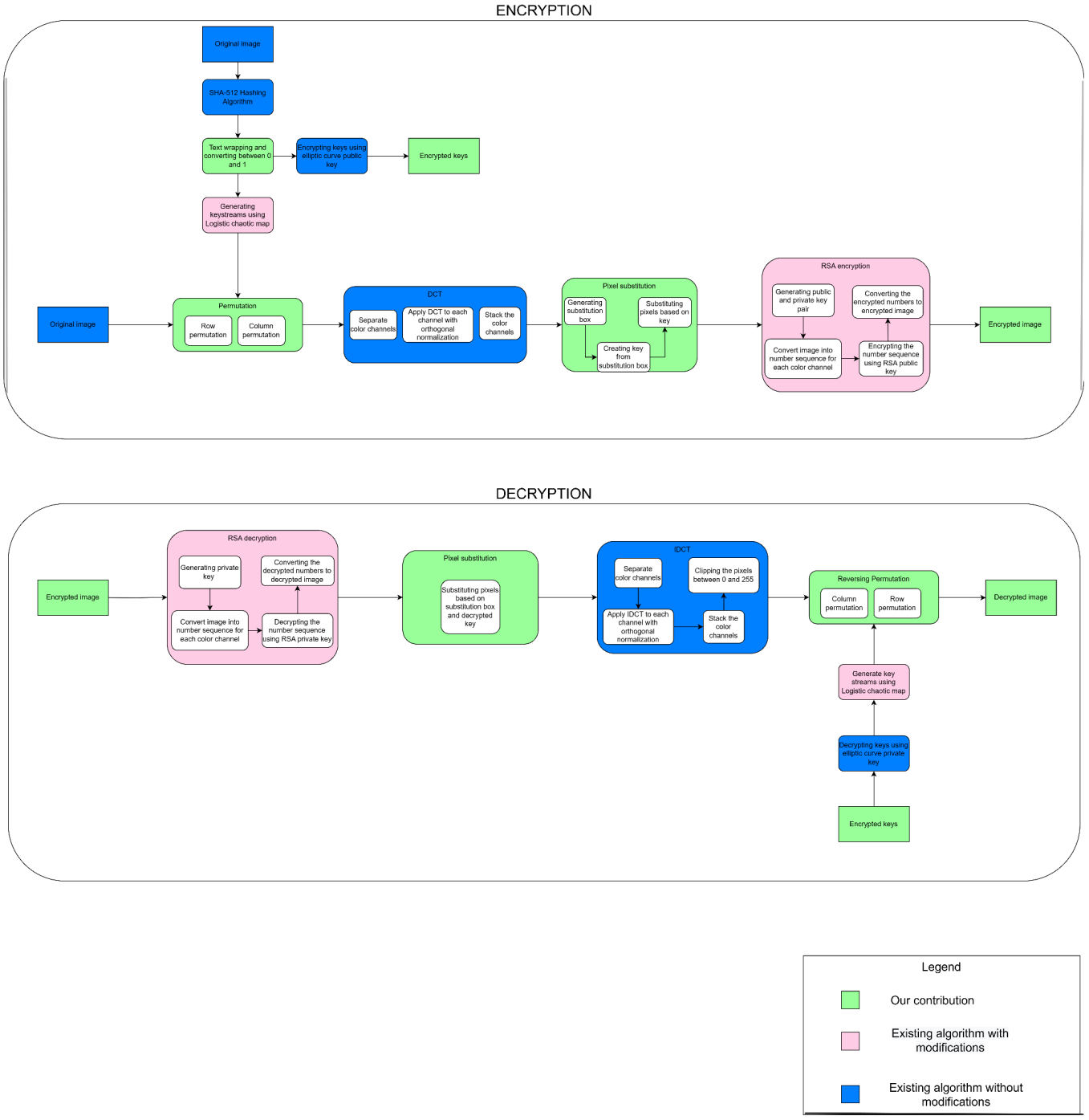


Figure 3.1 Overall architecture diagram

3.2.DETAILS OF MODULES

The complete framework of encryption decryption scheme can be divided into 10 main modules.

3.2.1 Complete Key generation

A hash value using SHA 512 algorithm from the original image is computed which can be seen in Figure 3.2.Then the hashed value is text wrapped and 4 parts of key are generated which is then converted between 0 and 1.The keys are encrypted using elliptic curve cryptography's public key which will be sent to receiver.Using logistic chaotic map keystreams are generated from non encrypted values which is shown in Pseudocode 3.1.

Input : Original Image

Output : Key streams and Encrypted keys

Pseudocode 3.1: Logistic chaotic map

```
Initialize r and x0
for i from 1 to N do
    x = r * x0 * (1 - x0) // Compute the next value of "x" using the logistic map equation
      x0 = x // Set the current value of "x" to the previous value for the next iteration
end for
```

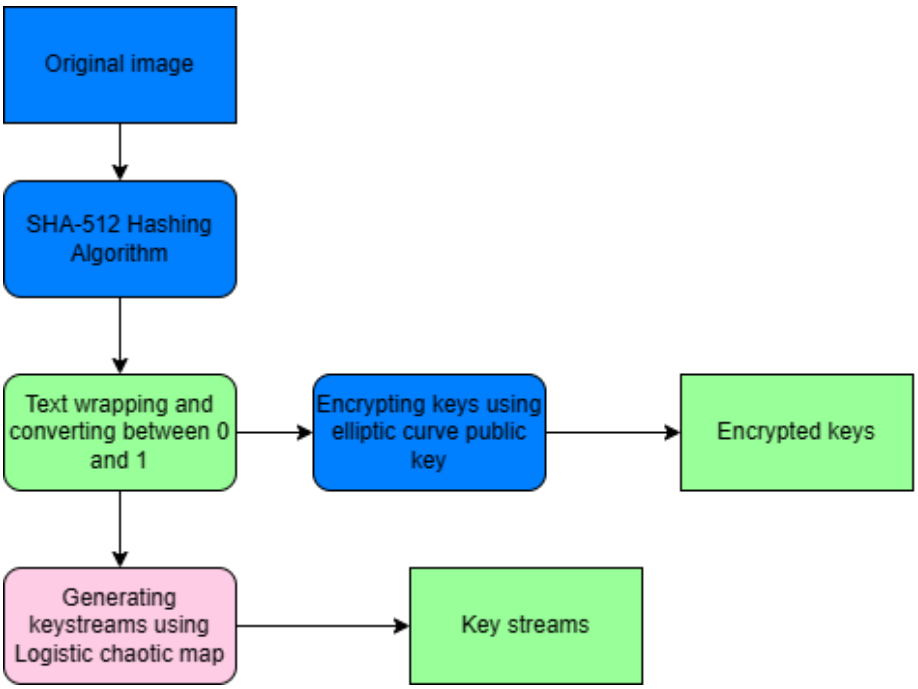


Figure 3.2 Key stream generation and Encryption

3.2.2 Permutation

The permutation shown in Figure 3.3 involves row permutation and then column permutation of the image.

3.2.2.1 Row Permutation

The image is converted into a numpy array.A permutation index array is generated using the keystream.The float array key is used to seed the random number generator ,a permutation index array is generated.Then the rows of the image array are permuted using the permutation index array.The permuted image array is converted back to an image and it is saved as written in Pseudocode 3.2.

Input : Original image

Output : Row permuted encrypted image

Pseudocode 3.2: Row Permutation of image

```
// Define the image dimensions
image_width = ...
image_height = ...
// Define the permutation order for the rows
permutation_order = ...
// Initialize a new image with the same dimensions as the original
new_image = create_image(image_width, image_height)
// Loop through each row in the original image, using the permutation order
for i = 0 to image_height - 1:
    // Get the index of the row to copy from the original image
    row_index = permutation_order[i]
    // Loop through each pixel in the row
    for j = 0 to image_width - 1:
        // Copy the pixel from the original image to the corresponding position in the new image
        new_image[j, i] = original_image[j, row_index]
// The new image now contains the original image with its rows permuted
```

3.2.2.2 Column Permutation

The Pseudocode 3.3 elaborates the following processes.The image is converted into a numpy array. The permutation index array based on the key array is generated.An index array is created which sorts the key in ascending order. This index array is applied to the columns of the image array using numpy's indexing.The permuted image array is converted back to an image and it is saved.

Input : Row permuted encrypted image

Output : Level 1 encrypted image

Pseudocode 3.3: Column Permutation of image

```
// Define the image dimensions
image_width = ...
image_height = ...
// Define the permutation order for the columns
permutation_order = ...
// Initialize a new image with the same dimensions as the original
new_image = create_image(image_width, image_height)
// Loop through each column in the original image, using the permutation order
for j = 0 to image_width - 1:
    // Get the index of the column to copy from the original image
    column_index = permutation_order[j]
    // Loop through each pixel in the column
    for i = 0 to image_height - 1:
        // Copy the pixel from the original image to the corresponding position in the new image
        new_image[j, i] = original_image[column_index, i]
// The new image now contains the original image with its columns permuted
```

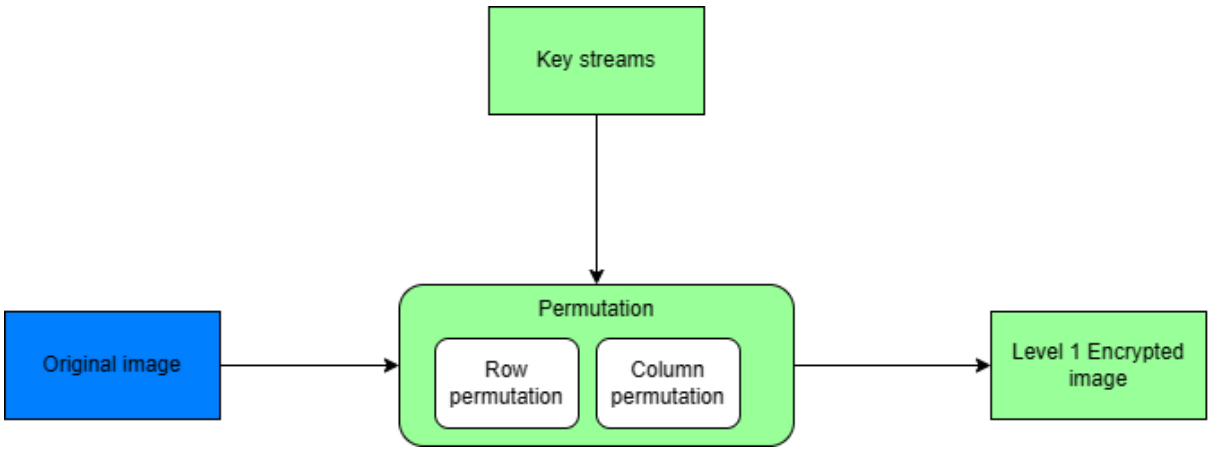


Figure 3.3 Permutation

3.3.3 DCT

The module diagram as shown in Figure 3.4 pictures the processes.The encrypted image from row permutation is taken as the input. The colour channels of the image are separated. Then DCT is applied to each colour

channel using orthogonal normalisation. Then the colour channels are stacked to get the DCT encrypted image written in Pseudocode 3.4.

Input : Level 1 encrypted image

Output : Level 2 Encrypted Image

Pseudocode 3.4: Discrete Cosine Transform(DCT)

```
// Initialize parameters
block_size = 8  // Set the block size
image = read_image()  // Read the image from a file

// Compute DCT for each block of pixels
for each block of size block_size x block_size in the image do
   // Compute the DCT for each coefficient in the block
   for u from 0 to block_size - 1 do
      for v from 0 to block_size - 1 do
         // Compute the sum for the current coefficient
         sum = 0
         for x from 0 to block_size - 1 do
            for y from 0 to block_size - 1 do
                  coefficient = image[x + block_x][y + block_y]  // Get the pixel
value for the current (x, y) position
                Cu = Cv = 1/sqrt(2)  // Set the constant factors to 1/sqrt(2) for u =
v = 0, and 1 otherwise
               if u == 0 then Cu = 1/sqrt(2)
               if v == 0 then Cv = 1/sqrt(2)
                 sum += Cu * Cv * coefficient * cos((2*x+1)*u*pi/(2*block_size))
* cos((2*y+1)*v*pi/(2*block_size))
         end for
         // Normalize and store the current DCT coefficient
         DCT_coefficients[u][v] = sum * (2/block_size) * (2/block_size)
      end for
   end for
   // Store the DCT coefficients for the current block in a file
   write_DCT_coefficients(DCT_coefficients)
end for
```

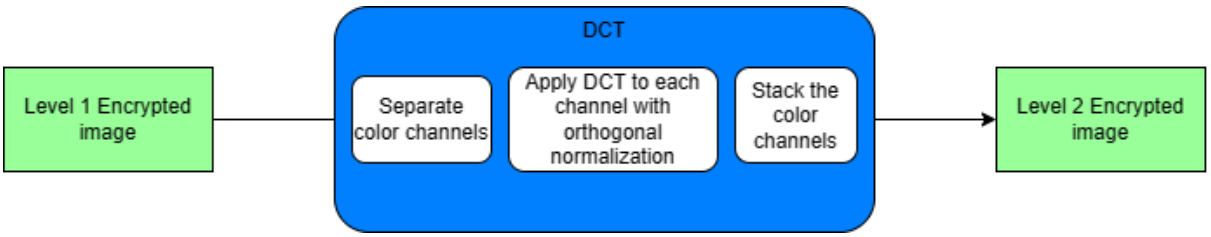


Figure 3.4 DCT

3.3.4 Pixel Substitution

The Pseudocode 3.5 puts forward the working of this module.A substitution based on the parameters of the image is generated. Then a random key is generated as the same size of the substitution box which is shown in Figure 3.5.The pixels are substituted based on keys.

Input : Level 2 Encrypted Image

Output : Level 3 Encrypted Image

Pseudocode 3.5: Pixel Substitution

```
image = read_image()  // Read the image from a file
key = generate_key()  // Generate a random key
substitution_box = generate_substitution_box()  // Generate a random substitution box
num_pixels = get_num_pixels(image)  // Get the number of pixels in the image

// Encrypt image by substitution using key and substitution box
for i from 0 to num_pixels - 1 do
   pixel = image[i]  // Get the current pixel
    new_pixel = substitute_pixel(pixel, substitution_box)  // Substitute the pixel using the substitution box
    encrypted_pixel = encrypt_pixel(new_pixel, key)  // Encrypt the substituted pixel using the key
   image[i] = encrypted_pixel  // Store the encrypted pixel in the image
end for
save_image(image, "encrypted_image.png")
save_substitution_box(substitution_box, "substitution_box.txt")
```

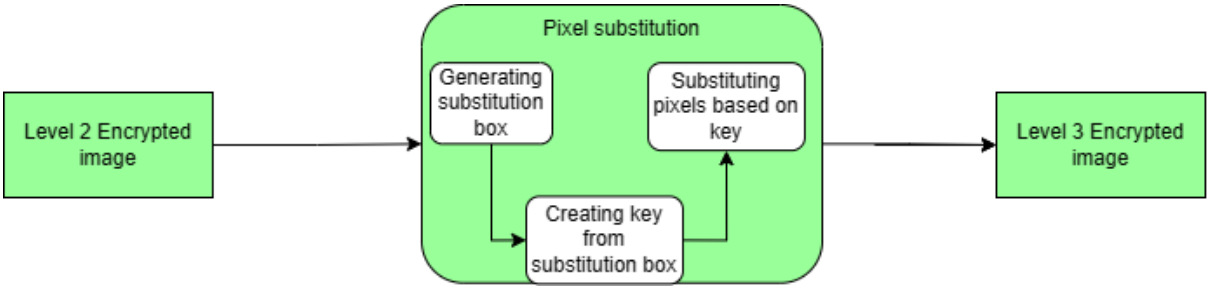


Figure 3.5 Pixel substitution

3.3.5 RSA Encryption

The steps involved in RSA encryption are written in Pseudocode 3.6. Public key and private key are generated using GCD method.Then the level 3 Encrypted Image is then converted into sequence of numbers.Then the number sequence is encrypted which has been clearly shown in Figure 3.6. This encrypted number sequence is converted back to image.

Input : Level 3 Encrypted Image

Output : Final Encrypted image

Pseudocode 3.6: RSA encryption

```
// Initialize the RSA keys
public_key, private_key = generate_RSA_keys()

// Load the image
image = load_image('image.png')

// Split the image into its color channels (e.g. Red, Green, Blue)
red_channel,green_channel, blue_channel = split_image_into_channels(image)

// Loop through each color channel
for channel in [red_channel, green_channel, blue_channel]:
    // Flatten the color channel to a 1D array of pixel values
    pixel_values = flatten_channel_to_array(channel)
```

```
    // Encrypt the pixel values using RSA
    encrypted_values = []
    for pixel_value in pixel_values:
        encrypted_value = RSA_encrypt(pixel_value, public_key)
        encrypted_values.append(encrypted_value)

        // Replace the original pixel values with the encrypted ones in the color channel
    replace_channel_with_encrypted_values(channel, encrypted_values)

// Merge the color channels back into a single image
encrypted_image = merge_channels_into_image(red_channel, green_channel, blue_channel)

// Save the encrypted image
save_image(encrypted_image, 'encrypted_image.png')
```

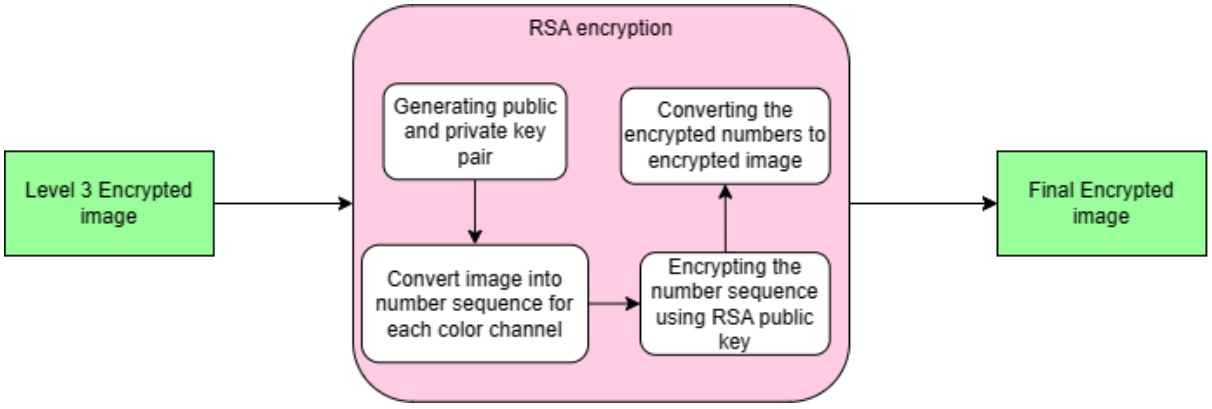


Figure 3.6 RSA Encryption

3.3.6 RSA Decryption

The decryption steps using RSA encryption has been given in Pseudocode 3.7. The private key is generated. The final encrypted image is converted into a sequence of numbers as shown in Figure 3.7. Then the number sequence is decrypted. The decrypted number sequence is converted back to image.

Input : Final encrypted image

Output : Level 1 Decrypted image

Pseudocode 3.7: RSA decryption

```
// Initialize the RSA keys
public_key, private_key = generate_RSA_keys()

// Load the image
image = load_image('image.png')

// Split the image into its color channels (e.g. Red, Green, Blue)
red_channel, green_channel, blue_channel = split_image_into_channels(image)

// Loop through each color channel
for channel in [red_channel, green_channel, blue_channel]:
    // Flatten the color channel to a 1D array of pixel values
    pixel_values = flatten_channel_to_array(channel)

    // Decrypt the pixel values using RSA
    decrypted_values = []
    for pixel_value in pixel_values:
        decrypted_value = RSA_decrypt(pixel_value, private_key)
        decrypted_values.append(decrypted_value)

    // Replace the original pixel values with the encrypted ones
    replace_channel_with_decrypted_values(channel, decrypted_values)

// Merge the color channels back into a single image
decrypted_image = merge_channels_into_image(red_channel, green_channel, blue_channel)

// Save the encrypted image
save_image(decrypted_image, 'decrypted_image.png')
```

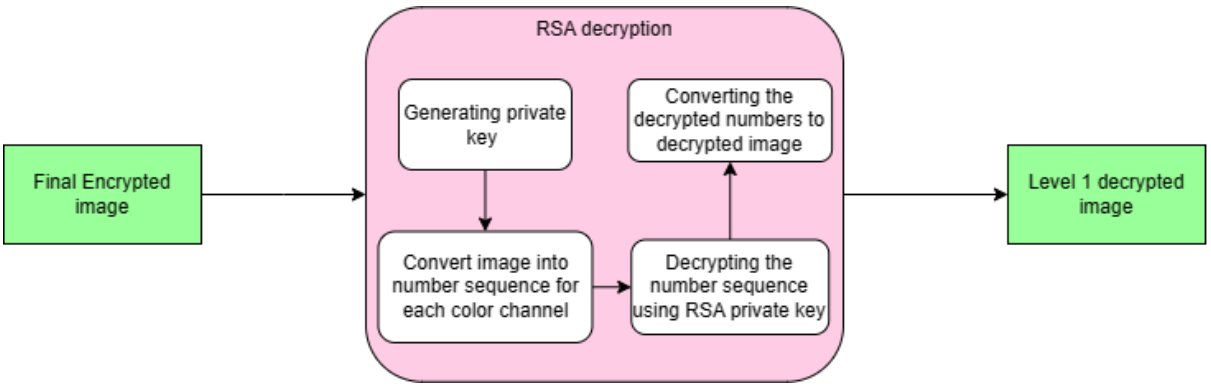


Figure 3.7 RSA Decryption

3.3.7 Reversing Pixel Substitution

The decrypted image from RSA is taken as indicated in Figure 3.8. The key for substitution is decrypted. Using the substitution box and the decrypted key the pixels are substituted to get the pixel substituted image.The Pseudocode 3.8 clearly gives the write up of above processes.

Input : Level 1 decrypted Image

Output : Level 2 decrypted Image

Pseudocode 3.8: Reversing Pixel Substitution

```
// Initialize parameters
image = read_image()  // Read the image from a file
num_pixels = get_num_pixels(image)  // Get the number of pixels in the image

// Decrypt image by substitution using key and substitution box
for i from 0 to num_pixels - 1 do
    pixel = image[i]  // Get the current pixel

        decrypted_pixel = decrypt_pixel(pixel, key)  // Decrypt the substituted pixel using the key
     new_pixel = substitute_pixel(decypted_pixel, substitution_box)  // Substitute the pixel using the substitution box
       image[i] = new_pixel  // Store the decrypted pixel in the image
end for
save_image(image, "decrypted_image.png") //save decrypted image to file
```

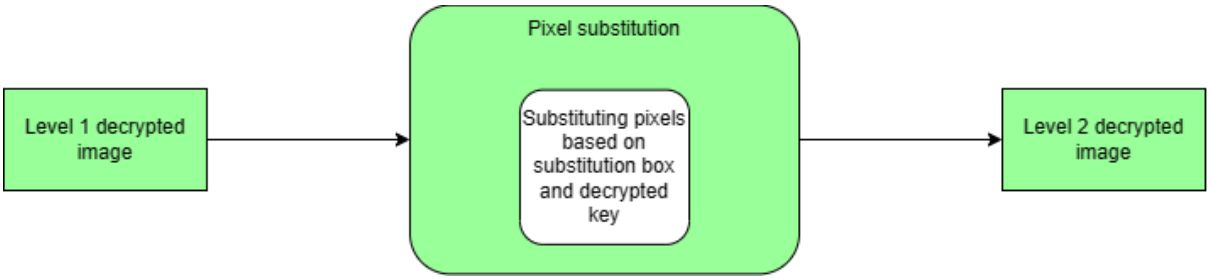


Figure 3.8 Reversing Pixel substitution

3.3.8 IDCT

The pixel substituted image is taken as the input.The colour channels are separated. Then for each channel IDCT is applied using orthogonal Normalisation as shown in Figure 3.9. The colour channels are stacked and pixels are clipped between 0 and 255 to get the decrypted image.The step by step writeup of working of this module is shown in Pseudocode 3.9.

Input : Level 2 decrypted image

Output : Level 3 decrypted Image

Pseudocode 3.9: Inverse Discrete Cosine Transform(IDCT)

```
// Initialize parameters
block_size = 8  // Set the block size
DCT_coefficients = read_DCT_coefficients()  // Read the DCT coefficients from a file

// Perform IDCT on each block of coefficients
for each block of DCT_coefficients do
    // Compute the IDCT for each pixel in the block
    for x from 0 to block_size - 1 do
        for y from 0 to block_size - 1 do
            // Compute the sum of DCT coefficients for the current pixel
            sum = 0
            for u from 0 to block_size - 1 do
```

```
            for v from 0 to block_size - 1 do
                Cu = Cv = 1/sqrt(2)  // Set the constant factors to 1/sqrt(2) for u = v = 0, and 1 otherwise
                if u == 0 then Cu = 1/sqrt(2)
                if v == 0 then Cv = 1/sqrt(2)
                sum += Cu * Cv * DCT_coefficients[u][v] * cos((2*x+1)*u*pi/(2*block_size)) * cos((2*y+1)*v*pi/(2*block_size))
            end for
            // Scale and round the sum to obtain the reconstructed pixel value
            pixel_value = round(sum * (2/block_size) + 128)
            // Clamp the pixel value to [0, 255]
            if pixel_value < 0 then pixel_value = 0
            if pixel_value > 255 then pixel_value = 255
            // Store the reconstructed pixel value
            reconstructed_block[x][y] = pixel_value
        end for
    end for
    // Store the reconstructed block in the image
    write_reconstructed_block(reconstructed_block)
end for
```

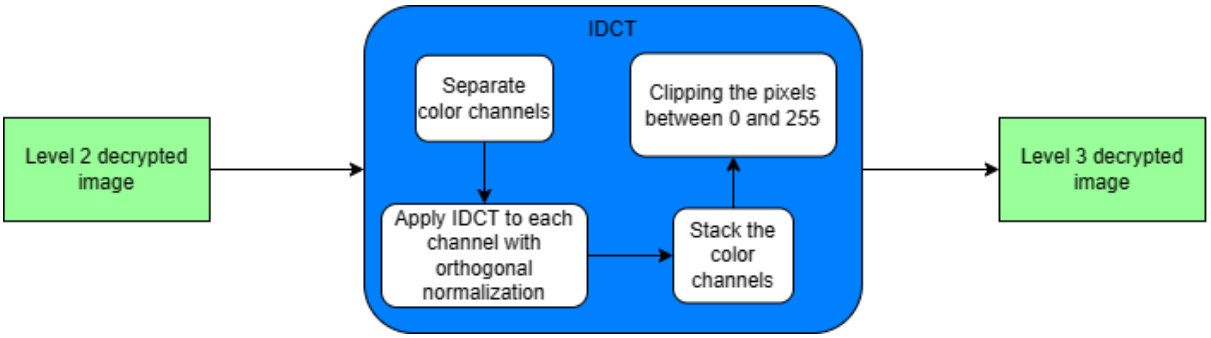


Figure 3.9 IDCT

3.3.9 Generating Key Streams

The step by step implementation is shown in Pseudocode 3.10. The keys are decrypted using elliptic curve cryptography's private key. Using these decrypted values as the input ,the logistic chaotic map as shown in Figure 3.10 generates the keystream.

Input : Encrypted keys

Output : Key streams

Pseudocode 3.10: Generating Key Streams

```
//decrypt key using elliptic curve cryptography
decrypted_value= bytes([encrypted_value[i] ^ key_hash[i % len(key_hash)]

//keystream using logistic chaotic map
Initialize r and x0
for i from 1 to N do
    x = r * x0 * (1 - x0) // Compute the next value of "x" using the logistic map equation
     x0 = x // Set the current value of "x" to the previous value for the next iteration
end for
```

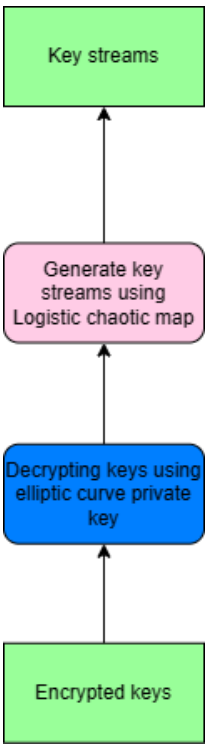


Figure 3.10 Generating Key streams

3.3.10 Reversing Permutation

In this module as shown in Figure 3.11, the image is first column permuted and then row permuted to get the final decrypted image.

3.3.10.1 Reverse Column Permutation

The image is converted into a numpy array. The permutation index array based on the key array is generated.An index array is created which sorts the key in ascending order as written in Pseudocode 3.11.This index array is applied to the columns of the image array using numpy's indexing.The permuted image array is converted back to an image and it is saved.

Input : Level 3 decrypted image

Output : Column permuted decrypted image

Pseudocode 3.11: Column Permutation for decryption

```
// Define the image dimensions
image_width = ...
image_height = ...
// Define the permutation order for the columns
permutation_order = ...
// Initialize a new image with the same dimensions as the original
new_image = create_image(image_width, image_height)
// Loop through each column in the original image, using the permutation order
for j = 0 to image_width - 1:
    // Get the index of the column to copy from the original image
    column_index = permutation_order[j]
    // Loop through each pixel in the column
    for i = 0 to image_height - 1:
        // Copy the pixel from the original image to the corresponding position in the new image
        new_image[j, i] = original_image[column_index, i]
// The new image now contains the original image with its columns permuted
```

3.3.10.2 Reverse Row Permutation

The overall implementation steps is written in Pseudocode 3.12.The image is converted into a numpy array.A permutation index array is generated using the keystream.The float array key is used to seed the random number generator ,a permutation index array is generated.Then the rows of the image

array are permuted using the permutation index array.The permuted image array is converted back to an image and it is saved.

Input : Column permuted decrypted image

Output : Final decrypted image

Pseudocode 3.12: Row Permutation for decryption

```
// Define the image dimensions
image_width = ...
image_height = ...
// Define the permutation order for the rows
permutation_order = ...
// Initialize a new image with the same dimensions as the original
new_image = create_image(image_width, image_height)
// Loop through each row in the original image, using the permutation order
for i = 0 to image_height - 1:
    // Get the index of the row to copy from the original image
    row_index = permutation_order[i]
    // Loop through each pixel in the row
    for j = 0 to image_width - 1:
        // Copy the pixel from the original image to the corresponding position in the new image
        new_image[j, i] = original_image[j, row_index]
// The new image now contains the original image with its rows permuted
```

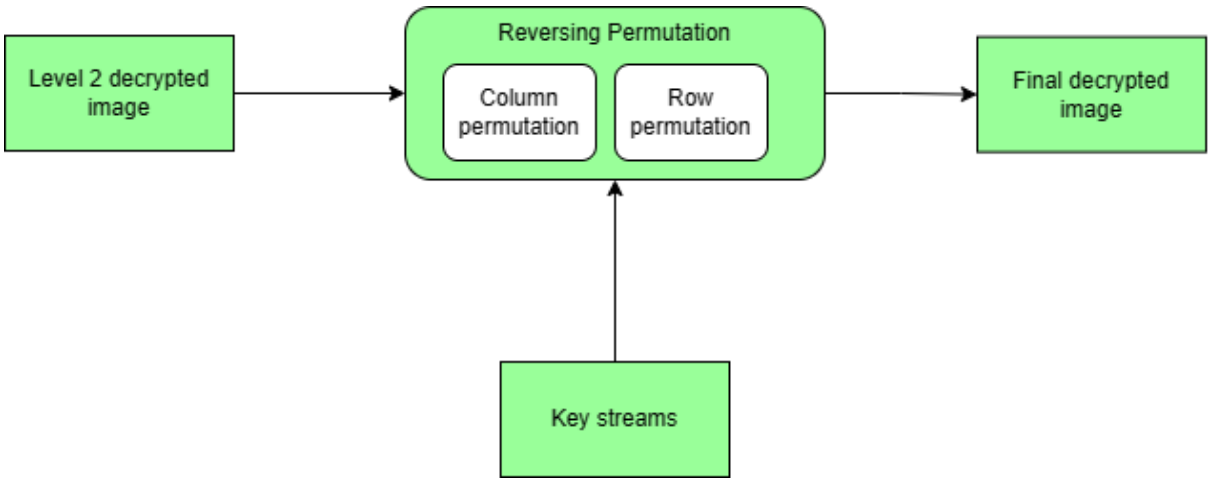


Figure 3.11 Reversing Permutation

CHAPTER 4
RESULTS AND DISCUSSION

4.1 DATASET DESCRIPTION

We use a Human faces dataset with pictures of around 7000 faces.It is a Male and Female Image coloured image dataset with special attention to senior citizens' pictures and a small portion of real-like fake faces to improve quality of encryption and decryption over a wide range of input pictures.The dataset includes images of various sizes.For testing our proposed system we have used coloured images of famous Indian personalities of size variation which includes Abdul Kalam, Kalpana chawla, Pratibha patel, M.S.Dhoni, A.R.Rahman and P.V.Sindhu.For comparative analysis we have taken standard test images used in respective compared research work like Baboon, Lena, Zelda, Pepper and Barbara.

4.2 RESULTS

The module-wise intermediate results are given below. Specific details of its implementation can be found at Appendix. The input will be a coloured image which is shown in Figure 4.1.

Input Image

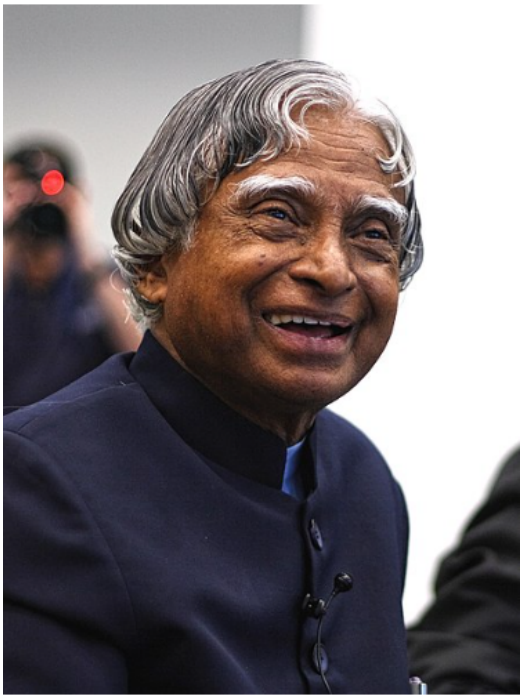


Figure 4.1 Abdul Kalam image

MODULE 1

The key streams that should be used for encryption are produced through a series of steps with methods like SHA-512,Text wrapping,Elliptic curve cryptography and Logistic chaotic maps.The first 10 values of the final keystreams as shown in Figure 4.2 is got through logistic chaotic map.

```
# Print the first 10 values of the key stream1
print(key_stream1[:10])
# Print the first 10 values of the key stream2
print(key_stream2[:10])
```

```
[595, 16, 64, 230, 566, 125, 396, 535, 226, 562]
[394, 538, 218, 554, 165, 479, 383, 550, 177, 498]
```

Figure 4.2 Key Streams of logistic chaotic map

MODULE 2

Row Permutation

From module 1 key streams obtained from logistic chaotic map is given as an output.From the original image an image array is generated and this array is row permuted with key stream 1 and the output as in Figure 4.3 will be a row permuted image.

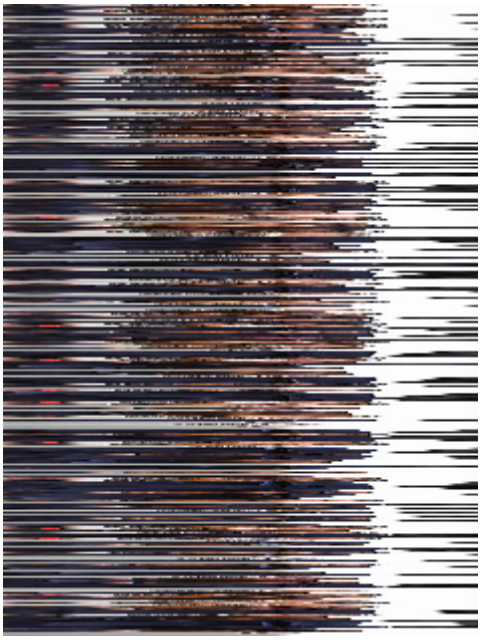


Figure 4.3 Row Permuted Image

Column Permutation

The row permuted image is taken as an input.From that image array is obtained.Using the keystream 2 the image array is permuted column wise.The output as in Figure 4.4 will be a column permuted image.

Figure 4.4 Column Permuted Image

MODULE 3

The input to this module will be a column permuted image.The colour channels of the image namely B,G,R are separated. Then DCT is applied to each colour channel performing a block-wise DCT using orthogonal normalisation. Then the colour channels are stacked to get the DCT encrypted image.The output which is shown in Figure 4.5 is a DCT encrypted image.

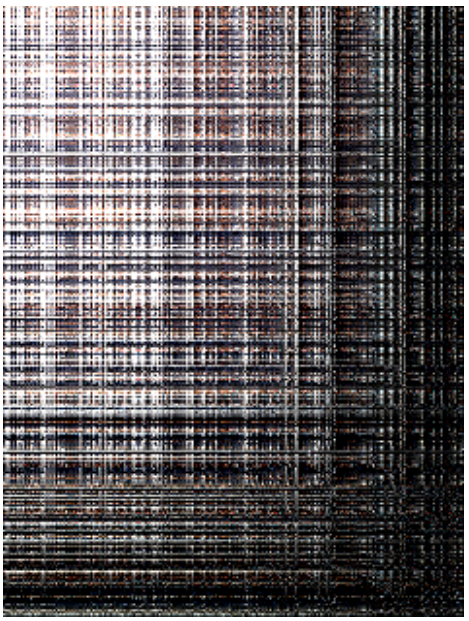


Figure 4.5 DCT Image

MODULE 4

A substitution box based on the parameters of the image is generated. Then a random key is generated as the same size of the substitution box.The pixels are substituted based on keys generating an image as shown in Figure 4.6 has its pixels substituted based on the key.

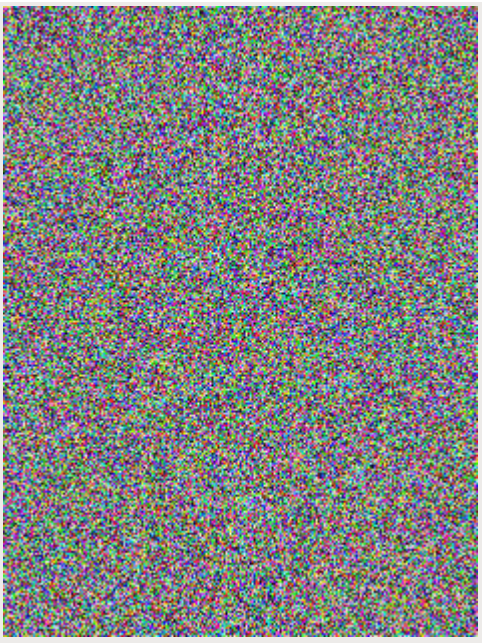


Figure 4.6 Pixel substituted Image

MODULE 5

Using RSA a pair of public and private key is generated.The pixel substituted Image is then converted into sequence of numbers.Then the number sequence is encrypted with the public key. This encrypted number sequence is converted back to image.This is the final encrypted image of the whole proposed scheme which is shown in Figure 4.7.

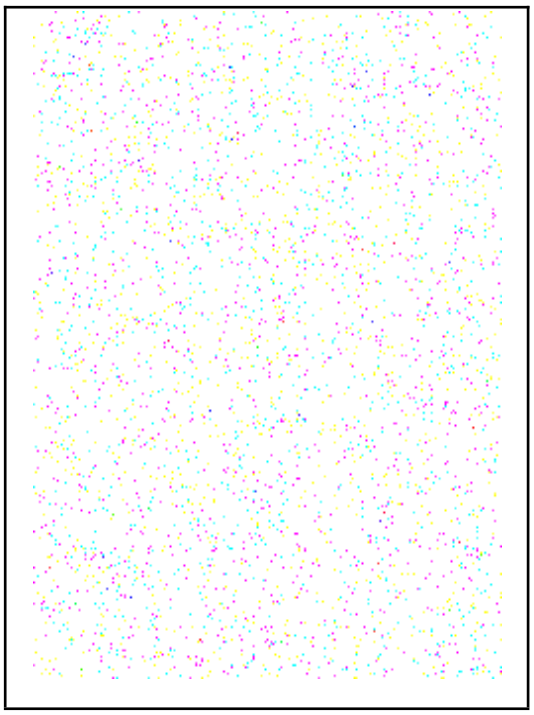


Figure 4.7 Final Encrypted Image

MODULE 6

The final encrypted image is converted into a sequence of numbers. Then the number sequence is decrypted using the private key. The decrypted number sequence is converted back to image and the output of this module as shown in Figure 4.8 will be a RSA decrypted image using the private key.

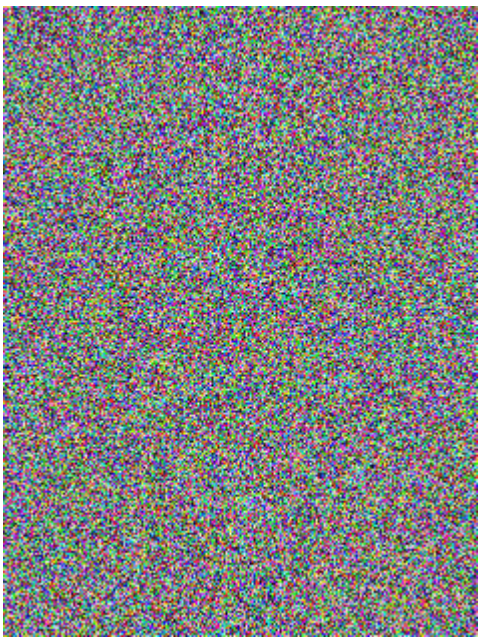


Figure 4.8 RSA decrypted image

MODULE 7

The decrypted image from RSA is taken as input producing an output of a re-substituted image as shown in Figure 4.9. The key for substitution is decrypted. Using the substitution box and the decrypted key the pixels are substituted to get the pixel substituted image.

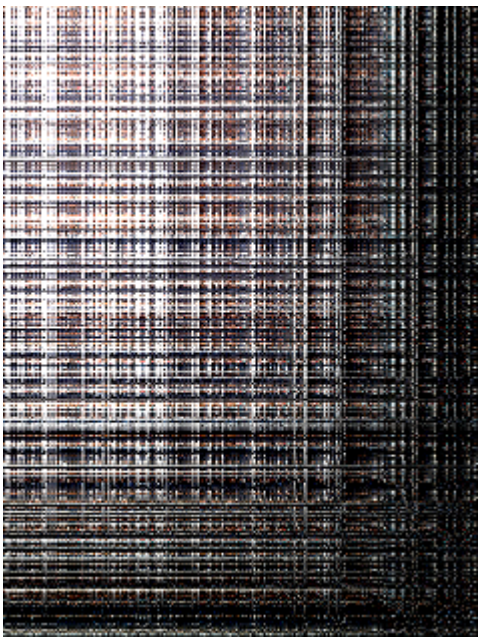


Figure 4.9 Reversed pixel substituted Image

MODULE 8

The colour channels are separated. Then for each channel IDCT performs a block-wise IDCT using orthogonal Normalisation. The colour channels are stacked and pixels are clipped between 0 and 255 to get the decrypted image which is shown in Figure 4.10.

Figure 4.10 IDCT Image

MODULE 9

The key streams as shown in Figure 4.11 which is to be used for decryption are produced using the private key and the encrypted key values passed to the sender.The encrypted keys will be decrypted using Elliptic curve cryptography and the key streams are generated using logistic chaotic map.

```
[32] # Print the first 10 values of the key stream1 using decrypted value
     print(d_key_stream1[:10])
     # Print the first 10 values of the key stream2 using decrypted value
     print(d_key_stream2[:10])

     [595, 16, 64, 230, 566, 125, 396, 535, 226, 562]
     [394, 538, 218, 554, 165, 479, 383, 550, 177, 498]
```

Figure 4.11 Keystreams from decrypted values

MODULE 10

Column Permutation

The key stream 2 is used for reverse permuting the IDCT image producing a column permuted image as shown in Figure 4.12.An array is obtained from the input image which is permuted column wise using the keystream.The permuted array will be converted back to image.The output which be column permuted image.

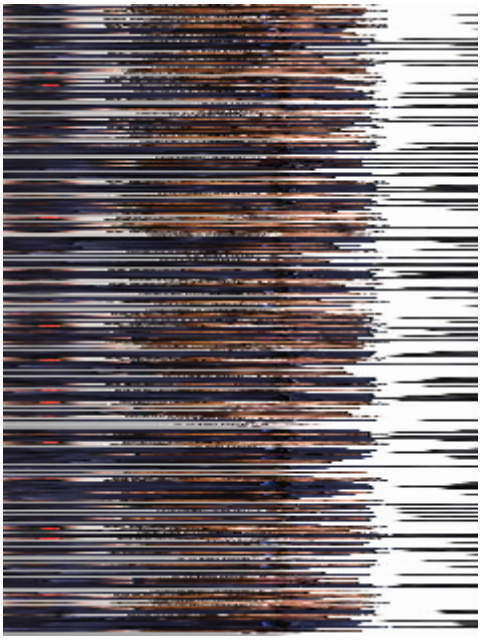


Figure 4.12 Reverse Column Permuted Image

Row Permutation

The column permuted image is taken as the input.An array is constructed from the image.The image array is row permuted using key stream 1.The permuted array is converted back to the image.The output image as shown in Figure 4.13 will be a row permuted image which is a final decrypted image.

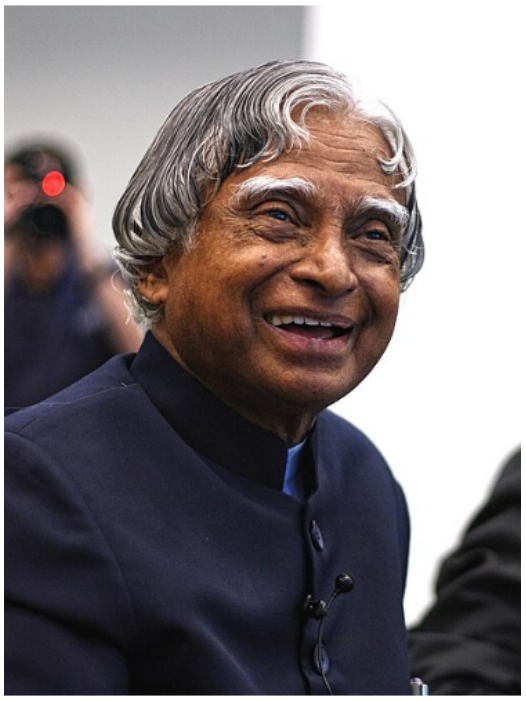


Figure 4.13 Final decrypted Image

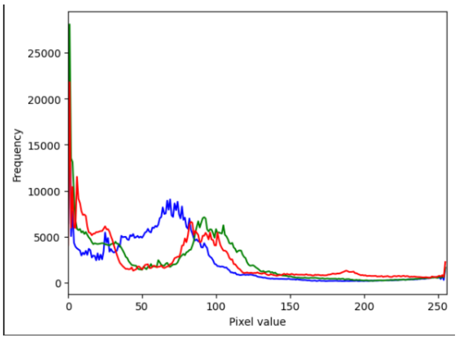
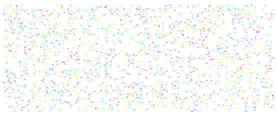
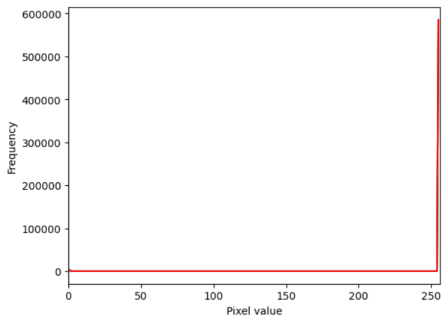
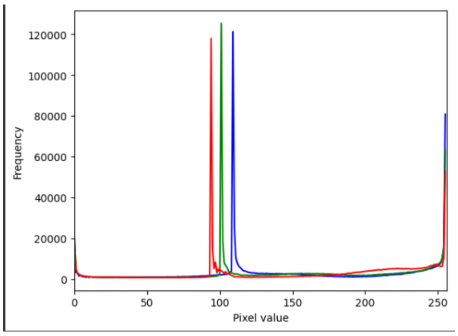
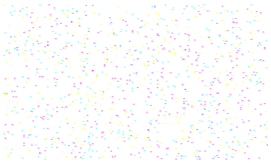
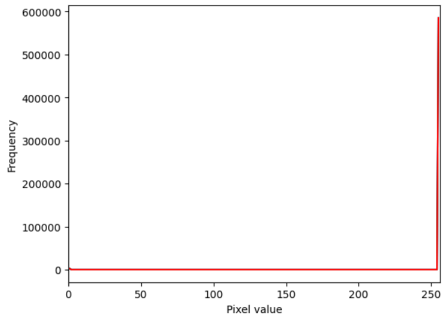
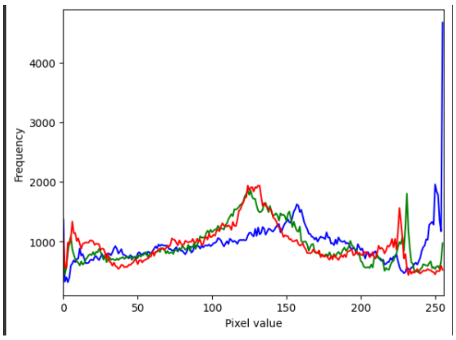
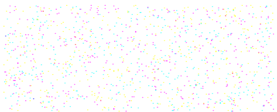
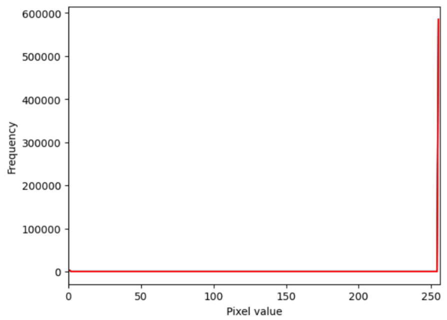
4.3 PERFORMANCE METRICS

We used the following metrics to evaluate the proposed encryption decryption scheme.

4.3.1. Histogram Analysis

It provides us with a graphical representation of the intensity distribution of an image.The histogram is a basic attribute of a digital image, which reflects the statistical characteristics of the relationship between image pixel level and image frequency.A good image encryption algorithm should make the pixels of the encrypted image more evenly distributed to effectively resist against statistical attacks.If the encrypted image has uniform histogram,it hides

information about the original image effectively.The histogram of encrypted image which is not uniform enough would capture information about the original image.The results of histogram analysis of our proposed scheme for the Kalpana chawla,Pratibha Patel,M S Dhoni,A R Rahman,P V Sindhu images are presented in Table 4.1.

Original image	Histogram of original image	Encrypted image	Histogram of encrypted image
			
			
			

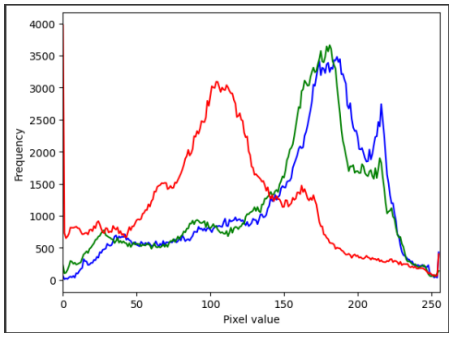
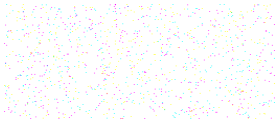
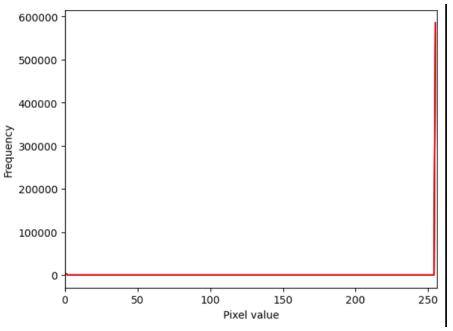
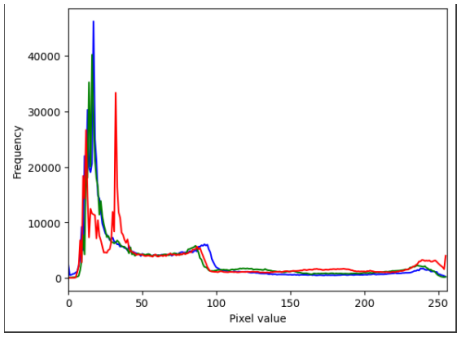
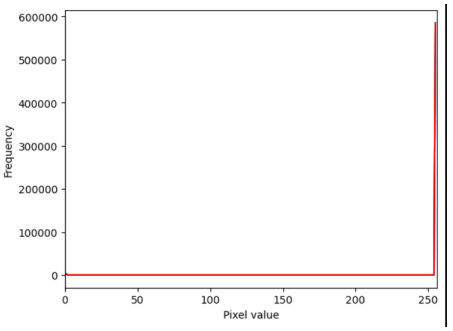
Original image	Histogram of original image	Encrypted image	Histogram of encrypted image
			
			

Table 4.1 Histogram analysis for original images and encrypted images

4.3.2. Correlation coefficient

CC is used to measure the association between two variables. In images, the pixels are highly correlated, and thus, image encryption algorithms mainly focus on reducing the association between the pixels of an encrypted image. The correlation coefficient is scaled between +1 and −1, where +1 indicates a maximum positive correlation, and −1 indicates a maximum negative correlation between two variables. Hence, if the correlation coefficient is equal to 0, it shows no association between the variables.The mathematical representation is written as follows

$$S = W \times H \qquad Eq.4.1$$

$$M(p) = \frac{1}{S}\sum_{i=1}^{S} p_i \qquad Eq.4.2$$

$$D(p) = \frac{1}{S}\sum_{i=1}^{S} (p_i - M(p)) \qquad Eq.4.3$$

$$cov(p_1, p_2) = \frac{1}{S}\sum_{i=1}^{S} (p_{1i} - M(p1))(p_{2i} - M(p2)) \qquad Eq.4.4$$

$$CC_{p1.p2} = \frac{cov(p_1, p2)}{\sqrt{D(p1)D(p2)}} \qquad Eq.4.5$$

W is the width of the image, H is the height of the image, S is the total number of pixels in the image.

p1i is the i th instance (pixel) of the plaintext image, and p2i is the i th instance (pixel) of the encrypted image.

M(p) is the mean of the pixel values, M(p1) is the mean of the pixel values of the plaintext image and M(p2) is the mean of the pixel values of the encrypted image.

D(p) is the variance of image, and D(p1) and D(p2) are the variance of the plaintext image and the cipher image, respectively;

p1 and p2 are the neighbouring pixels of the image.

cov(p1, p2) is the covariance of adjacent pixel.

Results as shown in Table 4.2 indicate that the proposed algorithm has small correlation values between the original and encrypted images.

Image	CC
Kalpana chawla	-0.00031
Pratibha Patel	0.00014
M S Dhoni	-0.00024
A R Rahman	0.00128
P V Sindhu	0.00019
AVG	0.00021

Table 4.2 Correlation coefficient b/w original and encrypted image

4.3.3 Key sensitivity analysis

A good image encryption scheme should be highly sensitive to the changes of key.So a minor change in the encryption key or the plain text would result in a completely different encrypted image. There will be great differences in the encrypted images with the changes of the initial value. Only the correct key can recover the original image. As a nutshell, the proposed image encryption scheme is sensitive to keys and can resist violent attacks.The correct decrypted key from elliptic curve cryptography with its decrypted image is shown in Table 4.3. The key with a slight variation in its decimal point with its decrypted image shows how much the scheme is sensitive in terms of key.

Key	Original image	Decrypted image
[0.6620373659022047, 0.564744967678191, 0.5833264301928258, 0.4719525592490519]		
[0.6720373659022048, 0.564744967678191, 0.5833264301928258, 0.4719525592490519]		

Table 4.3 Decryption using correct key and altered key

4.3.4.Horizontal,Vertical and Diagonal Correlation coefficient of adjacent pixels

A good encryption scheme should have minimal level of correlation between adjacent pixels. Here the horizontal,vertical and diagonal correlations of the encrypted images are analysed.In horizontal correlation, the correlation between the horizontal pixels of the image are computed. The correlation between the vertical pixels of the image are compared in case of vertical correlation and correlation between the diagonal pixels of the image are analysed for diagonal correlation. If the correlation value is 1 it means it is highly correlated and thus the image can be easily decrypted by attackers. On the other hand any value less than 0 shows minimum correlation between the adjacent pixels.As mentioned earlier, tests are performed on various images which are shown in Table 4.4 which present the results of the horizontal correlation, the vertical correlation and the diagonal correlation coefficients of the encrypted images. The proposed scheme has a smaller vertical correlation

coefficient, horizontal correlation coefficient, and diagonal correlation coefficient and is more resistant to attacks.

Image	HCC	VCC	DCC
Kalpana chawla	0.02708	0.02708	0.08124
Pratibha Patel	-0.00628	-0.00627	-0.01886
M S Dhoni	-0.00978	-0.00977	-0.02936
A R Rahman	-0.00664	-0.00663	-0.01993
P V Sindhu	-0.00905	-0.00904	-0.02716
AVG:	-0.00093	-0.00092	-0.00281

Table 4.4 HCC,VCC,DCC for encrypted images

4.3.5 Encryption time

The machine used to test the performance of the proposed algorithm has an Intel Core i5 CPU running at 3.5 GHz. The machine has 12 GB of RAM and runs the Windows 10 operating system. The proposed algorithm is run in Google colab. Table 4.5 shows the results of execution time of the algorithm on images with various sizes .The results indicate that the proposed algorithm has the fastest encryption speed.

Image	Size	Encryption time(s)
Kalpana chawla	1024 × 576	0.12
Pratibha Patel	800 × 1000	0.16
M S Dhoni	600 × 403	0.09
A R Rahman	720 × 405	0.10
P V Sindhu	1200 × 667	0.15
AVG:		0.12

Table 4.5 Encryption time of images

4.4 COMPARATIVE ANALYSIS

The comparison of the proposed scheme with various other research works has been shown in following sections.

4.4.1 Histogram analysis

The histogram of the encrypted images needs to be uniformly distributed.The histogram for the proposed scheme is more uniform than the histogram of the encrypted image using the Anees et al. [4] method, which shows that encrypted images using the proposed scheme hides more information and provides more resistance against attacks.

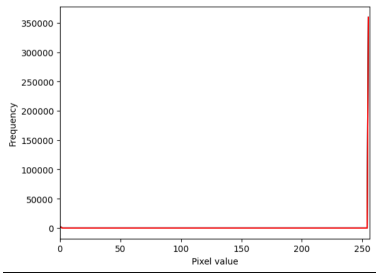
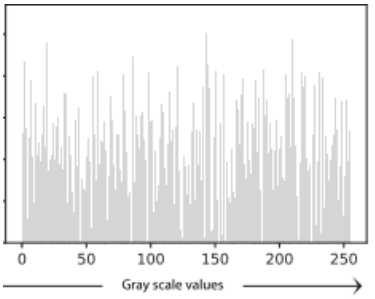
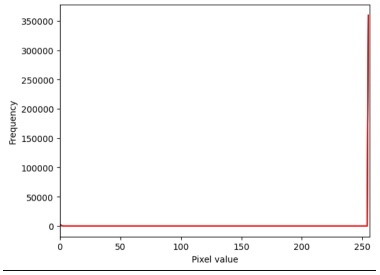
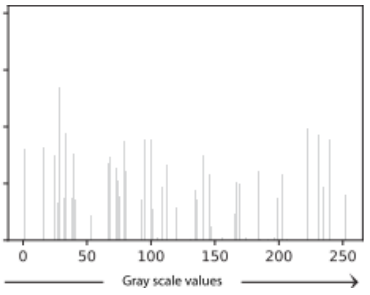
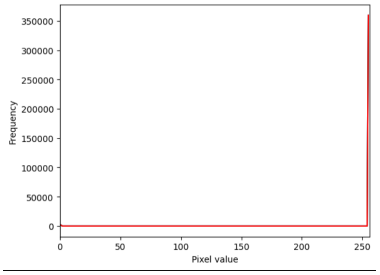
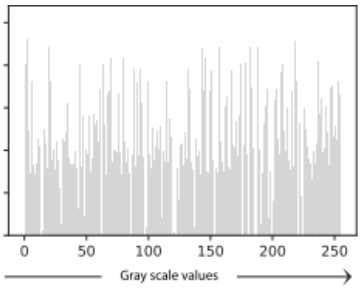
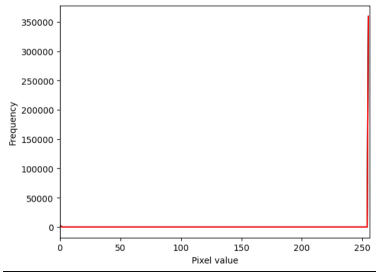
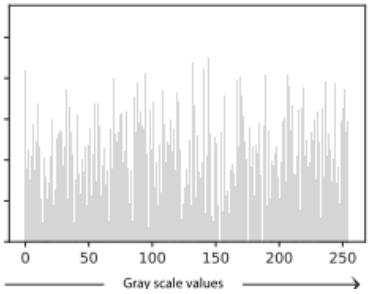
Image	Proposed scheme histogram	Anees et al.[4] histogram
Baboon		
Lena		
Zelda		
Pepper		

Table 4.6 Histogram of encrypted images of proposed scheme and Anees et al.[4]

4.4.2 Correlation coefficient

In correlation coefficient between the original image and the encrypted image in our proposed system is shown in Table.This CC is lower when compared to the avg CC obtained in Abdallah et al.[1] which is shown in Table 4.7.The CC of our proposed scheme being lower indicates that pixels of original images are highly changed through various steps of encryption scheme to make them less correlated. This makes it difficult to decrypt the image to get the original image thus protecting the images from being prone to attacks.

Correlation coefficient		
Images	Proposed scheme	Abdallah et al.[1]
Barbara	0.0001	0.0011
Baboon	-0.0006	0.0001
Lena	0.0019	0.0023

Table 4.7 CC b/w original and encrypted images of proposed scheme and Abdallah et al.[1]

4.4.3 Horizontal,Vertical and Diagonal Correlation coefficient of adjacent pixels

The Horizontal Correlation Coefficient[HCC],Vertical Correlation Coefficient[VCC],Diagonal Correlation Coefficient[DCC] represents the correlation between adjacent pixels of the encrypted images and it should be as minimal as possible. The average value of our scheme and some of the methods in the works of our references are shown in Table 4.8. The HCC,VCC,DCC of our proposed scheme when compared to others works shows that ours has a

lower value. This indicates that the adjacent pixels are highly uncorrelated in the encrypted images making it much more secure when compared to other works.

Literature	HCC	VCC	DCC
Alghamdi et al.[3]	0.0004	0.0005	−0.0004
Arif et al.[5]	0.00081	0.00061	0.03447
Hua et al.[9]	0.0023	-0.0086	0.0402
Lu et al.[11]	0.0007	−0.0004	0.0029
Alanezi et al.[2]	−0.0002	−0.0001	0.0011
Samiullah et al.[16]	-0.0357	-0.0357	-0.0223
Ge et al.[7]	-0.0004	0.00027	-0.00035
Proposed scheme	-0.00093	-0.00092	-0.00281

Table 4.8 HCC,VCC,DCC of encrypted images of proposed scheme and reference works

4.4.4 Encryption time

The encryption time of various test images have been found and their average is shown in Table 4.9. The encryption time of various other reference works is all shown in Table 4.9. Our scheme has more layers for encryption when compared to other works mentioned in references. Yet our scheme has a better speed of encryption. The proposed scheme puts forth a much secure at the same time faster encryption method for encrypting larger number images in a shorter time with much better level of security than other methods compared thereforth.

Literature	Encryption time(s)
Alghamdi et al.[3]	0.09
Arif et al.[5]	1.28
Hua et al.[9]	0.23
Lu et al.[11]	1.24
Alanezi et al.[2]	0.30
Samiullah et al.[16]	22.43
Ge et al.[7]	0.027
Proposed scheme	0.12

Table 4.9 Encryption time of proposed scheme and reference work

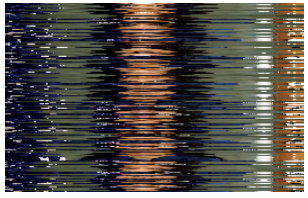
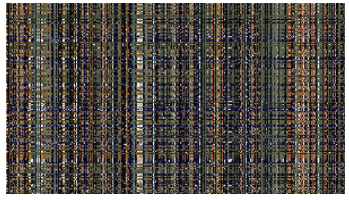
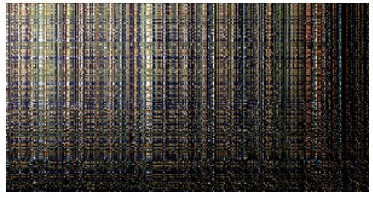
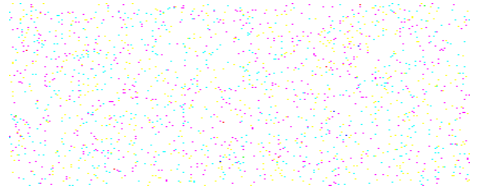
4.5 TEST CASES AND VALIDATION

The proposed system is tested with coloured images of 5 Indian famous personalities namely Kalpana chawla, Pratibha Patel, M.S.Dhoni, A.R.Rahman and P.V.Sindhu.These images vary in terms of height and width making the test case range in variety of heights and width.These images are the same set of image which were used for finding the evaluation metrics for the proposed scheme.The test images starting from the original image through undergoes various steps of encryption and decryption producing intermediate output images which is shown in Figure 4.5.1, 4.5.2, 4.5.3, 4.5.4, and 4.5.5.

Test case 1

Input

Dimension: 1024 × 576

Row permuted image	Column permuted image	DCT output image	Pixel substituted image	RSA Encrypted image(Final encrypted image)
				

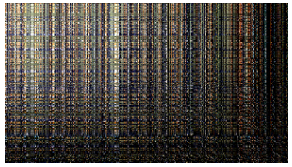
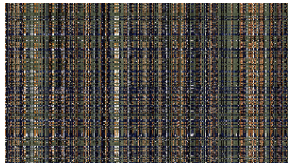
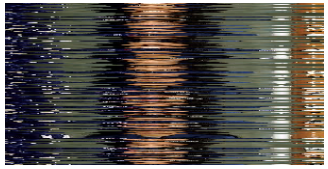
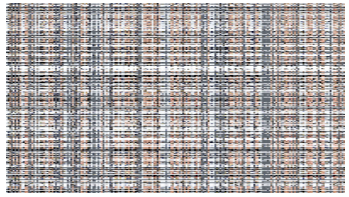
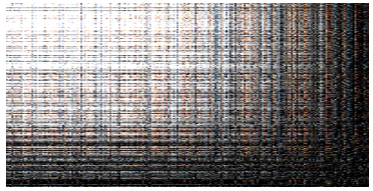
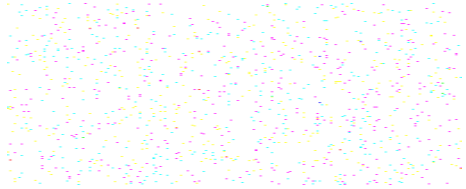
RSA decrypted image	Reverse Pixel substituted image	IDCT output image	Reverse column permuted image	Reverse row permuted image (obtained output)	Target output
					

Table 4.5.1 Kalpana Chawla testcase

Test case 2

Input

Dimension:800 × 1000

Row permuted image	Column permuted image	DCT output image	Pixel substituted image	RSA Encrypted image(Final encrypted image)
				

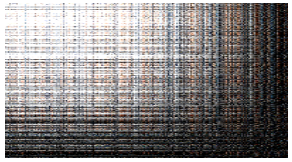
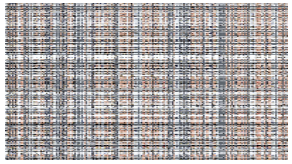
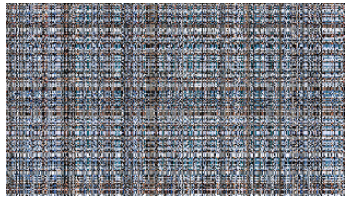
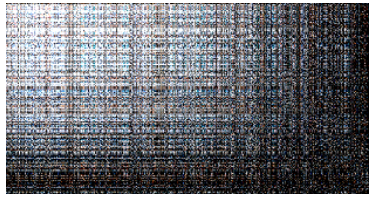
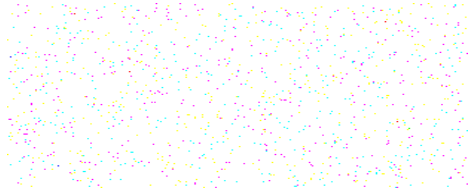
RSA decrypted image	Reverse Pixel substituted image	IDCT output image	Reverse column permuted image	Reverse row permuted image (obtained output)	Target output
					

Table 4.5.2 Pratibha Patil testcase

Test case 3

Input

 **Dimension:** 600 × 403

Row permuted image	Column permuted image	DCT output image	Pixel substituted image	RSA Encrypted image(Final encrypted image)
				

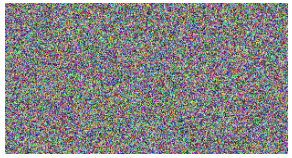
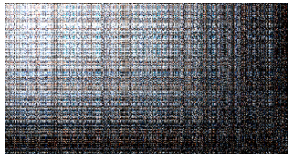
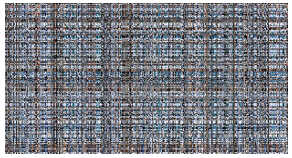
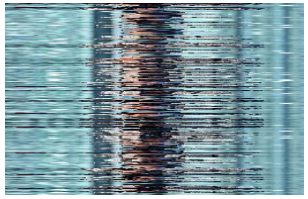
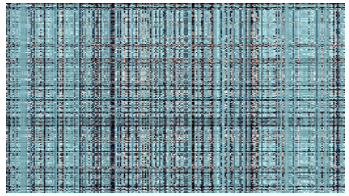
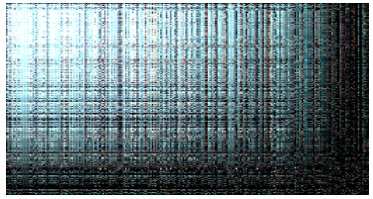
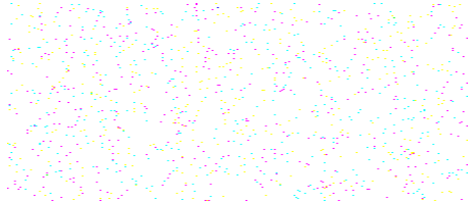
RSA decrypted image	Reverse Pixel substituted image	IDCT output image	Reverse column permuted image	Reverse row permuted image (obtained output)	Target output
					

Table 4.5.3 M.S.Dhoni testcase

Test case 4

Input

 **Dimension:** 720 × 405

Row permuted image	Column permuted image	DCT output image	Pixel substituted image	RSA Encrypted image(Final encrypted image)
				

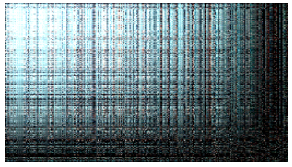
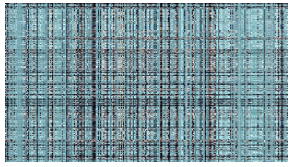
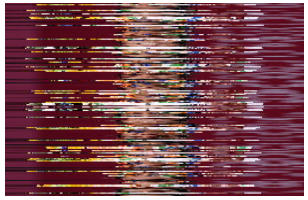
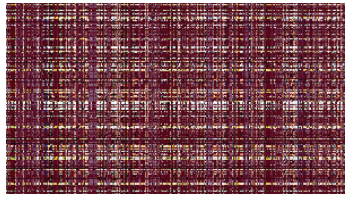
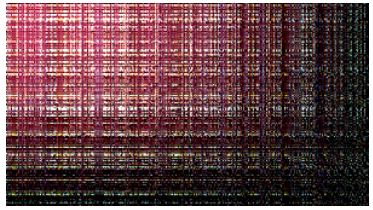
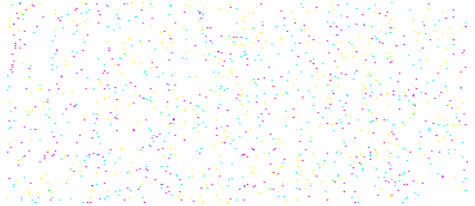
RSA decrypted image	Reverse Pixel substituted image	IDCT output image	Reverse column permuted image	Reverse row permuted image (obtained output)	Target output
					

Table 4.5.4 A.R.Rahman testcase

Test case 5

Input

Dimension: 1200 × 667

Row permuted image	Column permuted image	DCT output image	Pixel substituted image	RSA Encrypted image(Final encrypted image)
				

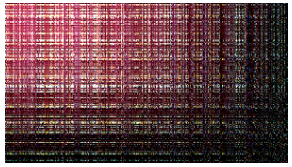
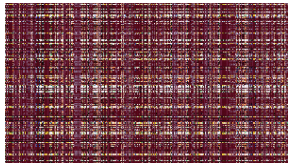
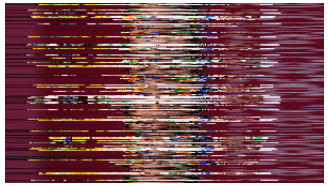
RSA decrypted image	Reverse Pixel substituted image	IDCT output image	Reverse column permuted image	Reverse row permuted image (obtained output)	Target output
					

Table 4.5.5 P.V.Sindhu testcase

CHAPTER 5
CONCLUSION AND FUTURE WORK

5.1 CONCLUSION

In this study, an image encryption scheme using logistic chaotic map and pixel substitution is proposed.The proposed scheme encrypts the image through layers of permutation,DCT,pixel substitution and RSA. Various keys which are encrypted through SAH-512, Elliptic curve cryptography, RSA encryption are used in various stages of the scheme to perform encryption and further used for decryption to get back the original image. The image through this scheme undergoes multiple levels of encryption thus offering more levels of security and randomness when compared to other state of art machines. The keys used in this scheme are highly sensitive in such a way that a small change in them while decryption would result in a completely different image.The correlation between the adjacent pixels in all dimension of horizontal,vertical and diagonal are minimised making the randomness much more high which makes it difficult to guess the original image when compared to other research works making the encryption more robust against statistical attacks.In terms of frequency of pixel distribution of encrypted image,the pixels are much more uniformly distributed when compared to other research works. Indeed, our proposed scheme has shown promising results in addressing the chosen plaintext attack with very high sensitivity towards the change in the key or the plaintext.

5.2 FUTURE WORKS

The proposed algorithm uses logistic maps for their minimal complexity and fast computation speed making it time efficient in encryption.Future works can explore other chaotic maps and include them to the proposed scheme. Further additional layers of encryption methods can be added to the existing layer to make it much more efficient.In addition, because of multiple layers

there is slight variation in decrypted image when compared to original image which affects the normalised correlation. Even though this is not much visible to human eyes,this problem could be further addressed in upcoming works.Furthermore, as a future work our proposed scheme can be extended for other media types, such as audio and video.

APPENDICES

A. IMPLEMENTATION DETAILS

MODULE-1

Original image

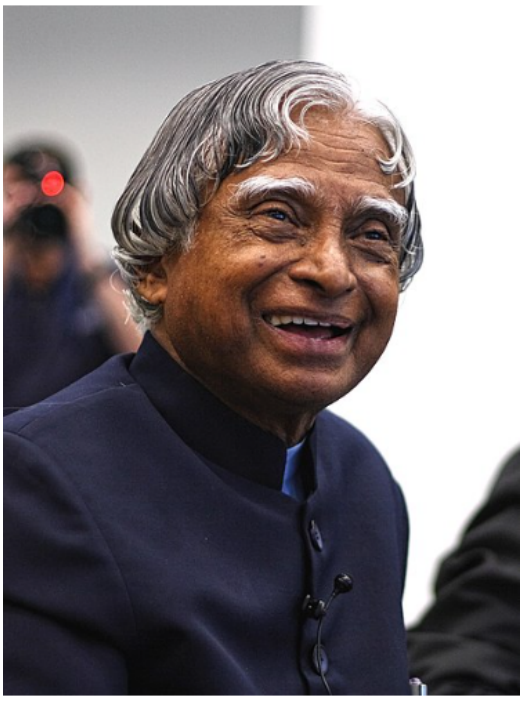


Figure A.1 Input image

a) SHA-512 Hashing Algorithm

Then the hash object is generated using SHA-512 algorithm.Then the hash object is updated with the image bytes and then it is changed to hexadecimal format using hexdigest() which is stored in the variable hash_hex.

```python
import hashlib
from PIL import Image

# Open the image file
image_file = Image.open("/content/A._P._J._Abdul_Kalam_in_2008.jpg")

# Convert the image to bytes
image_bytes = image_file.tobytes()

# Generate the hash object using SHA-512 algorithm
hash_object = hashlib.sha512()

# Update the hash object with the image bytes
hash_object.update(image_bytes)

# Generate the hash value in hexadecimal format
hash_hex = hash_object.hexdigest()
# Print the hash value
print("SHA-512 hash value:", hash_hex)
```

Figure A.2 SHA-512 Hashing Algorithm

Result

```
# Print the hash value
print("SHA-512 hash value:", hash_hex)

SHA-512 hash value: 0633fe8e4063dd434a706019bcc0c07f7b786bfda578615710ba23445e351797367b771d6a7b72678d37cea77977b354434b530c3f890b39c919f08ce765b3a9
```

Figure A.3 SHA-512 hash value

b) Text Wrapping and converting between 0 and 1

The code takes a hexadecimal string (hash_hex) and wraps it into multiple lines of text, with each line containing a maximum of 32 characters.

```python
#DIVIDING INTO 4 PARTS
#Our contribution
import textwrap
prekeys=textwrap.wrap(hash_hex, 32)
prekeys
```

Figure A.4 Text Wrapping

Result

```
['0633fe8e4063dd434a706019bcc0c07f',
 '7b786bfda578615710ba23445e351797',
 '367b771d6a7b72678d37cea77977b354',
 '434b530c3f890b39c919f08ce765b3a9']
```

Figure A.5 Text wrapped keys

Here each hexadecimal value in the prekeys list is converted to decimal, then modulus with 0.9999 is performed adding the resulting value to the keys list.

```
keys=[]
for i in prekeys:
  dec_value = int(i, 16)
  keys.append(dec_value%0.9999)
keys
```

Figure A.6 Converting between 0 and 1

Result

```
[0.466295806048032,
 0.20822832631649768,
 0.8424909894811603,
 0.42716115476829764]
```

Figure A.7 Processed keys of SHA-512

c) Encrypting keys using Elliptic curve cryptography

Initially, a private key is generated using the P-256 curve from the cryptography and the corresponding public key is also generated, and then converted into bytes. Finally,the resulting bytes are hashed using the SHA-256 algorithm from the hashlib module.

```
[5]  #ELLIPTIC CURVE KEY GENERATION
     import os
     import hashlib
     from cryptography.hazmat.primitives.asymmetric import ec
     from cryptography.hazmat.primitives.serialization import Encoding, PublicFormat

     # Generate a private key using P-256 curve
     private_key = ec.generate_private_key(ec.SECP256R1())

     # Get the corresponding public key
     public_key = private_key.public_key()

     # Convert the public key to bytes and hash it
     public_key_bytes = public_key.public_bytes(Encoding.X962, PublicFormat.CompressedPoint)
     key_hash = hashlib.sha256(public_key_bytes).digest()
```

Figure A.8 ECC Key generation

Each value in the keys list undergoes the following process.Initially,the values is converted into byte string using UTF-8 encoding.Then bitwise XOR is performed between each byte of the value and the corresponding byte of the key hash which is the final encrypted value for the corresponding input.

```
# Encrypting keys
en=[]
for i in keys:
  original_value = i
  value_bytes = bytes(str(original_value), 'utf-8')
  encrypted_value = bytes([value_bytes[i] ^ key_hash[i % len(key_hash)] for i in range(len(value_bytes))])
  en.append(encrypted_value)
  # Decrypt the encrypted value
  decrypted_value_bytes = bytes([encrypted_value[i] ^ key_hash[i % len(key_hash)] for i in range(len(encrypted_value))])
  decrypted_value = float(decrypted_value_bytes.decode('utf-8'))
  # Output the results
  print(en)
  print(encrypted_value)
```

Figure A.9 Key Encryption using ECC

Result

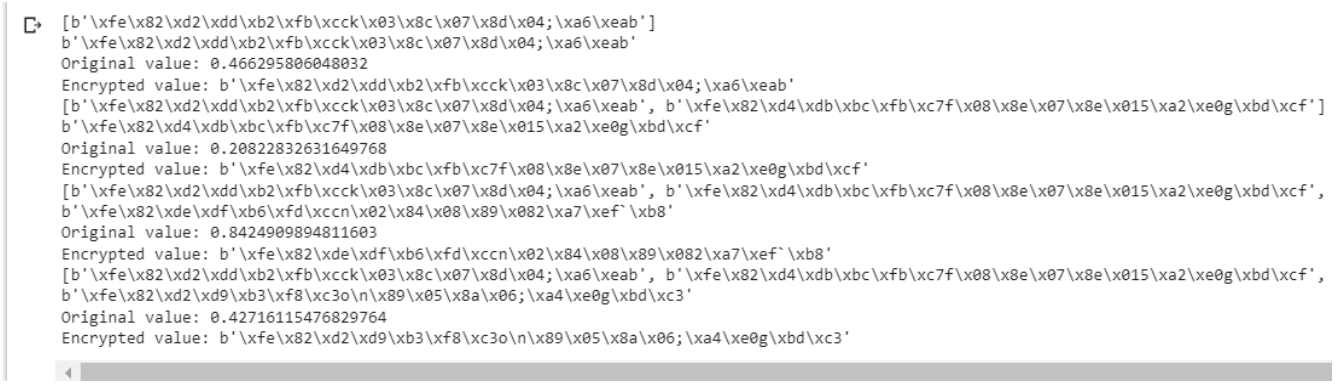


Figure A.10 Encrypted values of ECC

d) Generating Keystreams using Logistic Chaotic Map

This module is mainly used for generating keystreams that will be used to encrypt the image.First,the image to be encrypted is read and it is flattened into an one dimensional array and its parameters are stored in respective variables.Then,each pixel row in the image is traversed and by using the logistic map formula(r * x * (1 - x)),next value of x is found.further the keystream values are found by rounding x*(height-1) to the nearest integer.

```python
#GENERATING KEYSTREAMS USING LOGISTIC CHAOTIC MAP
import numpy as np
import cv2
image=cv2.imread('/content/A._P._J._Abdul_Kalam_in_2008.jpg')
flat_image = image.flatten()
height,width,c=image.shape

def logistic_map(x, r):
    """
    Generate key stream values using the logistic map.
    """
    key_stream = []
    for i in range(height):
        x = r * x * (1 - x)
        key_stream.append(int(np.round(x * height - 1))%height)
    return key_stream

# Set the initial value and parameters for the logistic map

r = 3.99

# Generate the key stream using the logistic map
key_stream1 = logistic_map(keys[0], r)
key_stream2 = logistic_map(keys[1], r)
key_stream3 = logistic_map(keys[2], r)
key_stream4 = logistic_map(keys[3], r)

# Print the first 10 values of the key stream1
print(key_stream1[:10])
```

Figure A.11 Keystream generation using logistic chaotic map

Result

```python
# Print the first 10 values of the key stream1
print(key_stream1[:10])
# Print the first 10 values of the key stream2
print(key_stream2[:10])
```

```
[595, 16, 64, 230, 566, 125, 396, 535, 226, 562]
[394, 538, 218, 554, 165, 479, 383, 550, 177, 498]
```

Figure A.12 Generated keystreams

MODULE-2

a)Row Permutation using generated keystream

In Row permutation,the first five values of the key_stream1 list are taken and used as a seed for a random number generator. Then a permutation of integers from 0 to num_rows - 1 is generated.The permutation indices will be used to shuffle the rows of the flattened image before encryption.The permutation_indices array is used to rearrange the rows of image_array in the same order as the permutation. Then the final permuted image is brought from the permuted image array.

```
[10] #GENERATING PERMUTATION SEQUENCE
     #Our contribution
     key = np.array(key_stream1[:5]) # replace with your float array key
     num_rows = image_array.shape[0]
     permutation_indices = np.random.RandomState(seed=key).permutation(num_rows)
```

```
[11] #ROW PERMUTATION
     #Our contribution
     permuted_image_array = image_array[permutation_indices, :]
```

```
[12] permuted_image = Image.fromarray(permuted_image_array)
     permuted_image.save('permuted_image.png')
```

Figure A.13 Row Permutation

Result

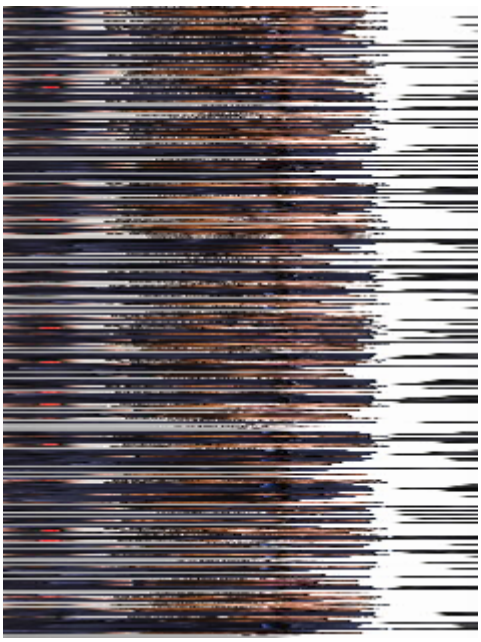


Figure A.14 Row permuted image

b)Column Permutation using generated keystream

In Column permutation,the first five values of the key_stream2 list are taken and used as a seed for a random number generator. Then a permutation of integers from 0 to num_cols - 1 is generated.The permutation_indices array is used to rearrange the columns of image_array in the same order as the permutation.Then the final permuted image is brought from the permuted image array.

```
[14] #GENERATING PERMUTATION SEQUENCE
     #Our contribution
     key = np.array(key_stream2[:5]) # replace with your float array key
     num_cols = image_array.shape[1]
     permutation_indices = np.random.RandomState(seed=key).permutation(num_cols)
```

```
[15] ##COLUMN PERMUTATION
     #Our contribution
     permuted_image_array = image_array[:,permutation_indices, :]
```

```
[16] permuted_image = Image.fromarray(permuted_image_array)
     permuted_image.save('col_permuted_image.png')
```

Figure A.15 Column Permutation

Result

Figure A.16 Column permuted image

MODULE-3

DCT

This DCT code applies the 2D Discrete Cosine Transform (DCT) to each colour channel (blue, green, and red) of the permuted image, using the OpenCV cv2.dct() function. The resulting DCT coefficients are then merged back into a single image.

```
[17] #DCT
     import cv2
     import numpy as np

     # Load image
     img = cv2.imread('col_permuted_image.png')

     # Split image into channels
     b, g, r = cv2.split(img)

     # Define DCT function
     def dct2(block):
         return cv2.dct(cv2.dct(block.T).T)



     # Perform DCT on each channel
     dct_b = dct2(b.astype(np.float32))
     dct_g = dct2(g.astype(np.float32))
     dct_r = dct2(r.astype(np.float32))

     dct_img = cv2.merge((dct_b,dct_g,dct_r))
     cv2.imwrite('dct_image.png', dct_img)
```

Figure A.17 DCT

Results

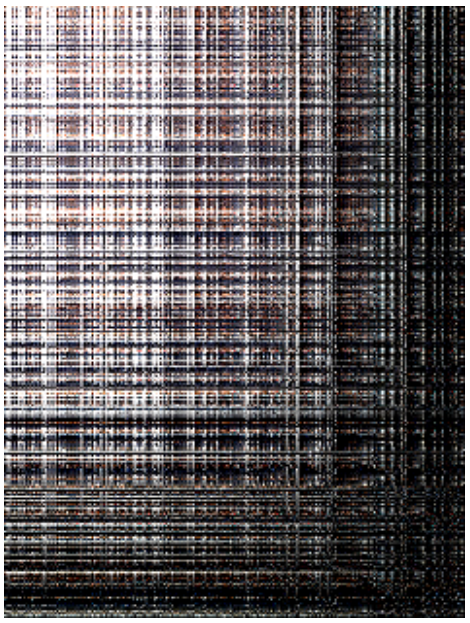


Figure A.18 DCT encrypted image

MODULE-4

a) Generating substitution box and key:

The code is generating a substitution box with a size equal to the number of pixels in the input image multiplied by the number of colour channels (RGB), and reshaping it to match the input image shape. It then generates a random key key with the same shape as the input image.The substitution box is used to encrypt the DCT coefficients by replacing each coefficient with the corresponding value in the substitution box using the key.

```python
#GENERATING SUBSTITUTION BOX
#Our contribution
import numpy as np
from PIL import Image

# Load the image data from a file
image = np.array(Image.open("dct_image.png"))

# Calculate the size of the substitution box needed
sbox_size = image.shape[0] * image.shape[1] * image.shape[2]

# Generate a random substitution box with the required size
sbox = np.arange(sbox_size, dtype=np.uint8)
np.random.shuffle(sbox)

# Reshape the substitution box to match the input image shape
sbox = sbox.reshape(image.shape)

# Generate a random key with the same shape as the input image
key = np.random.randint(256, size=image.shape).astype(np.uint8)

# Save the substitution box and key to files
np.save("sbox.npy", sbox)
arr1=np.load("sbox.npy")
print(arr1)
np.save("key.npy", key)
arr2=np.load("key.npy")
print("keys:")
print(arr2)
```

Figure A.19 Substitution box

Results

```
[[[250 251 166]
  [ 82  89 175]
  [137 196 190]
  ...
  [ 39 169  35]
  [103 109 133]
  [132  81  78]]

 [[151  38 141]
  [156  36  86]
  [131 213   2]
  ...
  [149 179  14]
  [ 85 153  78]
  [231 115 152]]

 [[ 11 193  61]
  [107 205  36]
  [ 34  10  66]
  ...
  [ 45  15   1]
  [ 38 224 246]
  [ 92 248 153]]

 ...

 [[220 160 120]
  [ 28 111  74]
  [139  80 122]
  ...
  [146 230  14]
  [ 36 172 126]
  [ 36 194  63]]

 [[103 219 163]
  [193 116   1]
  [212 245 199]
```

Figure A.20 Generated substitution box

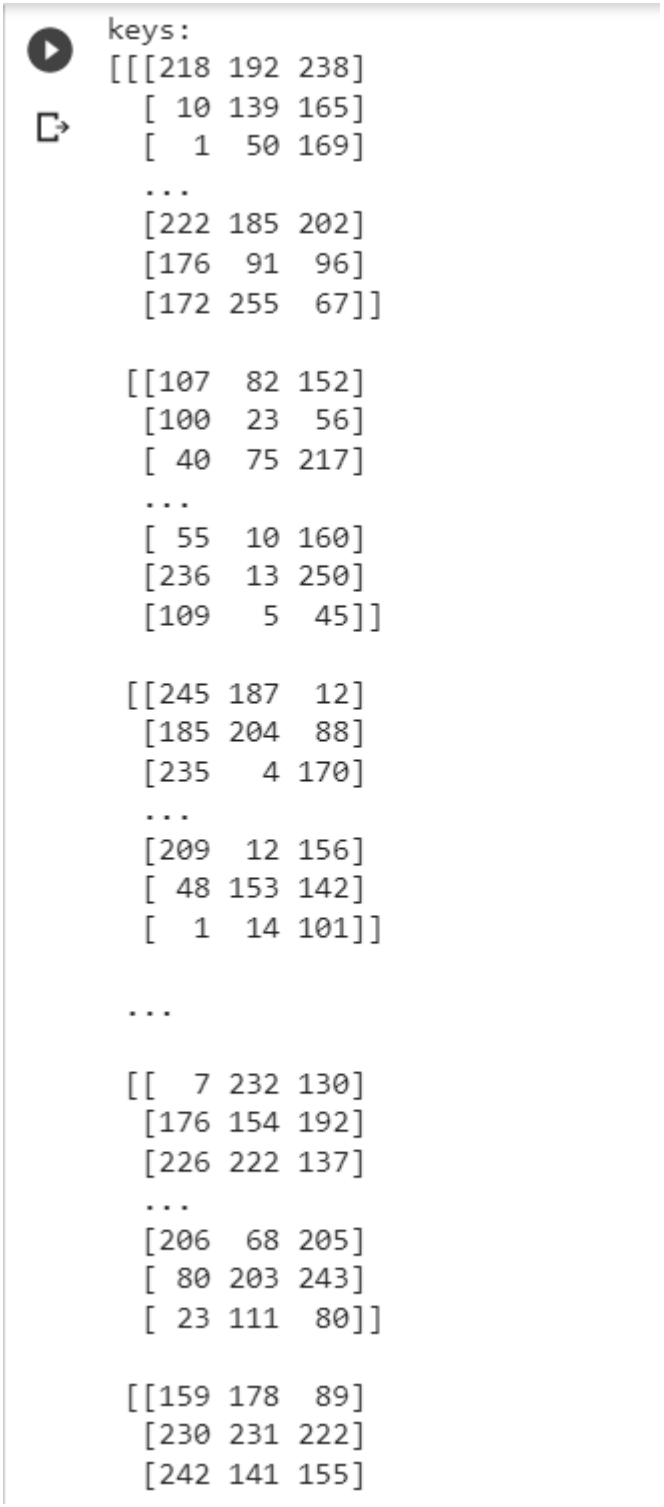


Figure A.21 Generated substitution key

b) Substituting pixels using key

Here,the DCT image produced earlier are converted into respective numpy array, on which bitwise XOR operation is done and the output will be the substituted image.

```
[19] #PERFORMING BITWISE XOR
     #Our contribution
     imgforps = Image.open("/content/dct_image.png")
     img_array = np.array(imgforps)
     encrypted_array = np.bitwise_xor(img_array, arr2)
     encrypted_img = Image.fromarray(encrypted_array)
     encrypted_img.save("sub_encrypted_image.png")
```

Figure A.22 Pixel substitution

Result

Figure A.23 Pixel substituted image

MODULE-5

RSA Encryption

The generate_keypair function takes two prime numbers p and q as input and returns a tuple of the public and private keys. The encrypt function takes a

public key and a message as input and returns the encrypted message as a list of numbers.Then the image is then converted into sequence of numbers using image_to_numbers.Then the number sequence is encrypted. This encrypted number sequence is converted back to image using numbers_to_image function.

```python
import random
from PIL import Image

# Generate public and private keys
def generate_keypair(p, q):
    n = p * q
    phi = (p - 1) * (q - 1)
    e = random.randrange(1, phi)
    while gcd(e, phi) != 1:
        e = random.randrange(1, phi)
    d = modinv(e, phi)
    return ((n, e), (n, d))

# Greatest common divisor
def gcd(a, b):
    while b != 0:
        a, b = b, a % b
    return a

# Modular inverse
def modinv(a, m):
    g, x, y = egcd(a, m)
    if g != 1:
        raise Exception('Modular inverse does not exist')
    return x % m

# Extended Euclidean algorithm
def egcd(a, b):
    if a == 0:
        return (b, 0, 1)
    else:
        g, y, x = egcd(b % a, a)
        return (g, x - (b // a) * y, y)
```

Figure A.24 RSA Encryption

```python
# Convert sequence of numbers to image
def numbers_to_image(numbers, width, height, image_path):
    img = Image.new('RGB', (width, height))
    pixels = img.load()
    i = 0
    for y in range(height):
        for x in range(width):
            r = numbers[i]
            g = numbers[i + 1]
            b = numbers[i + 2]
            pixels[x, y] = (r, g, b)
            i += 3
    img.save(image_path)

# Encrypt a message using a public key
def encrypt(public_key, message):
    n, e = public_key
    encrypted = [pow(char, e, n) for char in message]
    return encrypted
```

```python
[21] # Example usage
p = 179
q = 2789
public_key, private_key = generate_keypair(p, q)
image_path = '/content/sub_encrypted_image.png'
numbers, width, height = image_to_numbers(image_path)
encrypted_numbers = encrypt(public_key, numbers)
numbers_to_image(encrypted_numbers, width, height, 'final_encrypted_img.png')
```

Figure A.24 RSA Encryption

Result

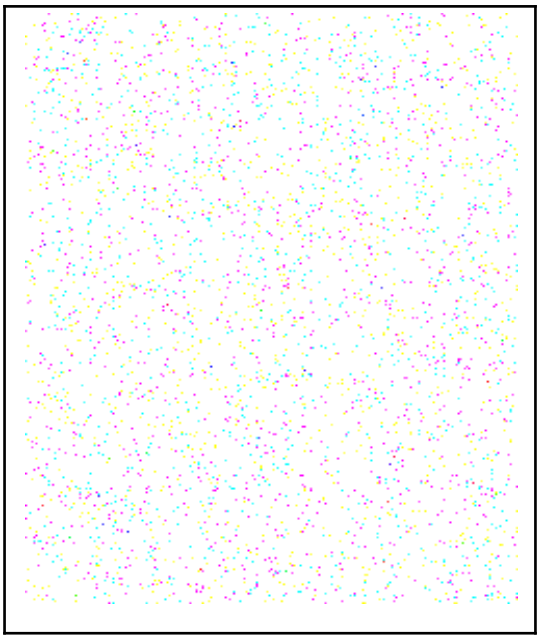


Figure A.25 Final Encrypted image

MODULE-6

RSA Decryption

The decryption function takes a private key and encrypted image as input.The decryption is performed using modular exponentiation with the private key parameters n and d. The private key is assumed to be a tuple of the form (n, d). The encrypted message is assumed to be a list of integers representing the encrypted characters of the message.

```
[22] ##DECRYPTION
```

```
[23] #RSA decryption
```

```
[24] def decrypt(private_key, message):
         n, d = private_key
         decrypted = [pow(char, d, n) for char in message]
         return decrypted
```

```
[25] dimage_path = '/content/final_encrypted_img.png'
     numbersd1, widthd1, heightd1 = image_to_numbers(dimage_path)
     decrypted_numbers = decrypt(private_key, encrypted_numbers)
     numbers_to_image(decrypted_numbers, widthd1, heightd1, 'rsa_decrypted_img.png')
```

Figure A.26 RSA Decryption

Result

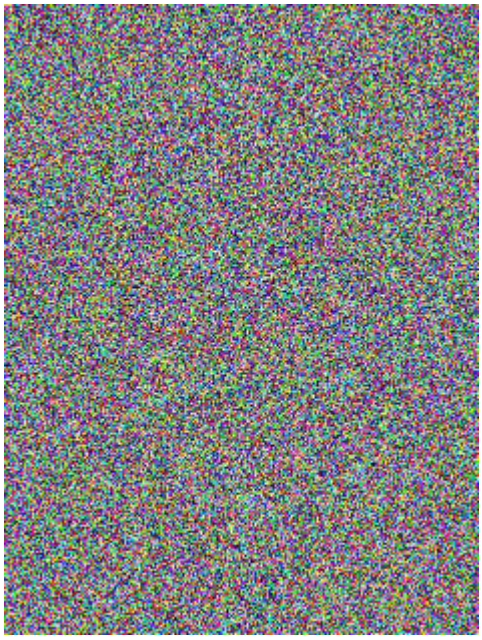


Figure A.27 RSA decrypted image

MODULE-7

Reversing Substitution

After RSA decryption,the image is converted into a numpy array and then a bitwise XOR is performed.And then from the decrypted array,the original image (partially decrypted) will be formed.

```
[26] #pixel substitution on rsa decrypted image
     #Our contribution
     deimgforps = Image.open("/content/rsa_decrypted_img.png")
     deimg_array = np.array(deimgforps)
     decrypted_array = np.bitwise_xor(deimg_array, arr2)
     decrypted_img = Image.fromarray(decrypted_array)
     decrypted_img.save("decrypted_image_from_ps.png")
```

Figure A.28 Reversing substitution

Result

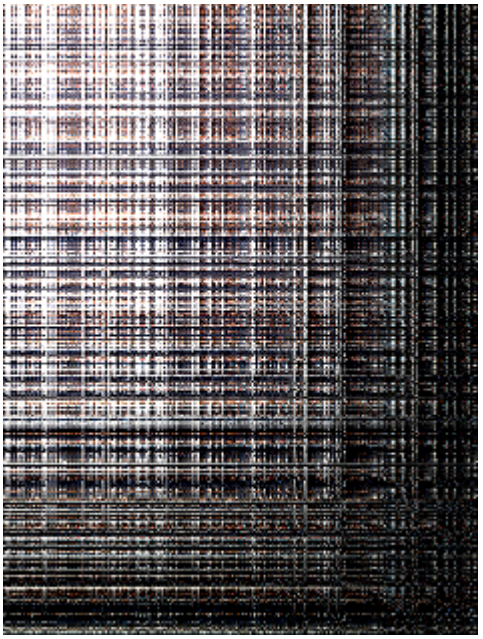


Figure A.29 Reverse substituted image

MODULE-8

IDCT

First,the decrypted image is split into its individual color channels (blue, green, and red), and the inverse discrete cosine transform (IDCT) to each channel using a custom function idct2() is approved. The resulting channels are merged back together into an RGB image, and the image is saved as idct_image.png.

```
[27] #IDCT on decyrpted image grom pixel substitution
     import cv2
     import numpy as np

     # Load image
     img = cv2.imread('decrypted_image_from_ps.png')

     # Split image into channels
     b, g, r = cv2.split(img)

     # Define IDCT function
     def idct2(block):
         return cv2.idct(cv2.idct(block.T).T)

     idct_b = idct2(dct_b).astype(np.uint8)
     idct_g = idct2(dct_g).astype(np.uint8)
     idct_r = idct2(dct_r).astype(np.uint8)

     idct_img = cv2.merge((idct_b, idct_g, idct_r))
     cv2.imwrite('idct_image.png', idct_img)
```

Figure A.30 IDCT

Result

Figure A.31 IDCT decrypted image

MODULE-9

a) Decrypting key using elliptic key cryptography

The code is iterating through each encrypted value in the list and performing a bitwise XOR operation with a key hash value, which is obtained by taking the modulo of the length of the key hash. The values are then converted to bytes and decoded into a float value, which is appended to the dkeys list.

```
#DECRYPTING KEYS USING ELLIPTIC CURVE CRYPTOGRAPHY
en_len = len(en)
dkeys=[]
print(en_len)
for i in range(en_len):
  a=en[i]
  decrypted_value_bytes = bytes([a[i] ^ key_hash[i % len(key_hash)] for i in range(len(a))])
  decrypted_value = float(decrypted_value_bytes.decode('utf-8'))
  dkeys.append(decrypted_value)
  # Output the results
  print(f"decrypted value: {decrypted_value}")
print(dkeys)
```

Figure A.32 Decrypting key using elliptic curve cryptography

Result

```
4
decrypted value: 0.466295806048032
decrypted value: 0.20822832631649768
decrypted value: 0.8424909894811603
decrypted value: 0.42716115476829764
[0.466295806048032, 0.20822832631649768, 0.8424909894811603, 0.42716115476829764]
```

Figure A.33 Decrypted keys from ECC

b) Generating Keystreams using Logistic Chaotic Map

This module is mainly used for generating keystreams that will be used to decrypt the image.First,the image to be decrypted is read and it is flattened into an one dimensional array and its parameters are stored in respective variables.Then,each pixel row in the image is traversed and by using the logistic

map formula(r * x * (1 - x)),next value of x is found.further the keystream values are found by rounding x*(height-1) to the nearest integer.

```
[29] #keystream generation using logistic chaotic map
```

```
[30] d_image=cv2.imread('/content/idct_image.png')
```

```
[31] d_flat_image = d_image.flatten()
     d_height,d_width,d_c=d_image.shape

     def d_logistic_map(x, r):
         """
         Generate key stream values using the logistic map.
         """
         d_key_stream = []
         for i in range(d_height):
             x = r * x * (1 - x)
             d_key_stream.append(int(np.round(x * d_height - 1))%d_height)
         return d_key_stream

     # Set the initial value and parameters for the logistic map

     r = 3.99

     # Generate the key stream using the logistic map
     d_key_stream1 = d_logistic_map(dkeys[0], r)
     d_key_stream2 = d_logistic_map(dkeys[1], r)
     d_key_stream3 = d_logistic_map(dkeys[2], r)
     d_key_stream4 = d_logistic_map(dkeys[3], r)

     # Print the first 10 values of the key stream
     print(d_key_stream1[:10])
```

Figure A.34 Keystream generation using logistic chaotic map

Result

```
[32] # Print the first 10 values of the key stream1 using decrypted value
     print(d_key_stream1[:10])
     # Print the first 10 values of the key stream2 using decrypted value
     print(d_key_stream2[:10])

     [595, 16, 64, 230, 566, 125, 396, 535, 226, 562]
     [394, 538, 218, 554, 165, 479, 383, 550, 177, 498]
```

Figure A.35 Keystreams of decrypted value

MODULE-10

a) Reversing column permutation

This module involves generating a random order for the columns of the image array using a portion of a key stream passed as a seed to numpy.random.RandomState().Then a re-permutation indices is generated which is then inverted.Then the permutation is done by indexing an array with re_permutation_indices and then inverse_indices are used to reorder the columns to their original order.Finally,column shuffling is done on the numpy array using the inverse permutation indices.These indices can be converted into decrypted image.

```
[34] #reversing column permutation on image got from IDCT
     #Our contribution

[35] dp_image = Image.open('/content/idct_image.png')
     dp_image_array = np.array(dp_image)

[36] dkey = np.array(d_key_stream2[:5]) # replace with your float array key
     num_cols1 = dp_image_array.shape[1]
     re_permutation_indices = np.random.RandomState(seed=dkey).permutation(num_cols1)

[37] inverse_indices = np.argsort(re_permutation_indices)

[38] re_permuted_image_array = dp_image_array[:,inverse_indices, :]

[39] re_permuted_image = Image.fromarray(re_permuted_image_array)
     re_permuted_image.save('reverse_col_permuted_image.png')
```

Figure A.36 Reverse column permutation

Result

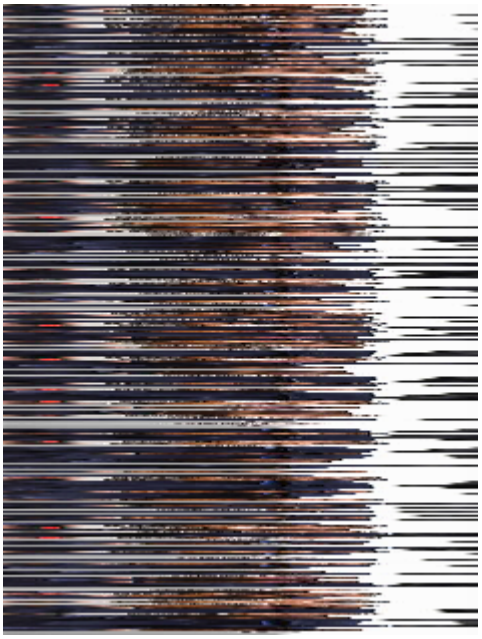


Figure A.37 Reverse column permuted image

b) Reversing row permutation

The number of rows is obtained from the shape of the input image array. A portion of key stream1 is seed to the numpy.random.RandomState() method, which returns a random number generator with the seed set to the specified value. This generator generates a permutation of the row indices using the permutation() method which is stored in the variable re_permutation_indices.Then the permutation is done by indexing an array with re_permutation_indices and then inverse_indices are used to reorder the rows to their original order.Finally,row shuffling is done on the numpy array using the inverse permutation indices.These indices can be converted into the final decrypted image.

```
[40] #reversing row permutation
     #Our contribution
```

```
[41] dp_image = Image.open('/content/reverse_col_permuted_image.png')
     dp_image_array = np.array(dp_image)
```

```
[42] dkey1 = np.array(d_key_stream1[:5]) # replace with your float array key
     num_rows1 = dp_image_array.shape[0]
     re_permutation_indices = np.random.RandomState(seed=dkey1).permutation(num_rows1)
```

```
[43] inverse_indices = np.argsort(re_permutation_indices)
```

```
[44] re_permuted_image_array = dp_image_array[inverse_indices, :]
```

```
[45] re_permuted_image = Image.fromarray(re_permuted_image_array)
     re_permuted_image.save('final_decrypted_image.png')
```

Figure A.38 Reverse row permutation

Result

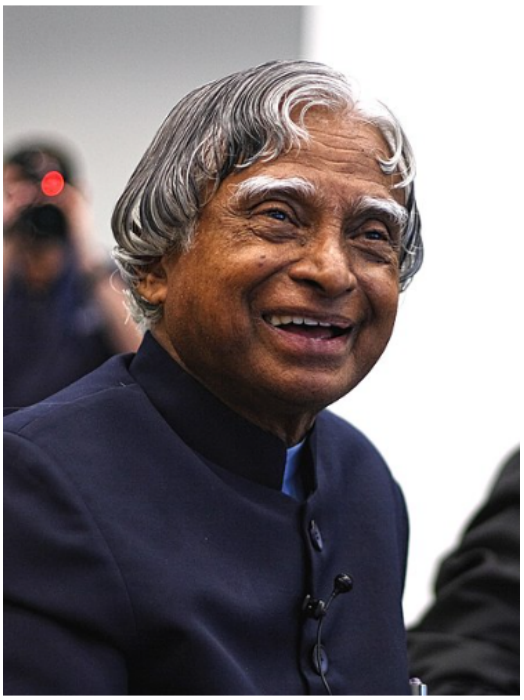


Figure A.39 Final decrypted image

REFERENCES

[1] Abdallah, A. A., & Farhan, A. K. (2022). A new image encryption algorithm based on multi chaotic system. *Iraqi Journal of Science*, 324–337.

[2] Alanezi, A., Abd-El-Atty, B., Kolivand, H., Abd El-Latif, A. A., Abd El-Rahiem, B., Sankar, S., & S. Khalifa, H. (2021). Securing digital images through simple permutation-substitution mechanism in cloud-based Smart City Environment. *Security and Communication Networks*, *2021*, 1–17.

[3] Alghamdi, Y., Munir, A., & Ahmad, J. (2022). A lightweight image encryption algorithm based on chaotic map and random substitution. *Entropy*, *24*(10), 1344.

[4] Anees, A., Siddiqui, A., & Ahmed, F. (2014). Chaotic substitution for highly autocorrelated data in encryption algorithm. *Communications in Nonlinear Science and Numerical Simulation*, *19*(9), 3106–3118.

[5] Arif, J., Khan, M. A., Ghaleb, B., Ahmad, J., Munir, A., Rashid, U., & Al-Dubai, A. Y. (2022). A novel chaotic permutation-substitution image encryption scheme based on logistic map and random substitution. *IEEE Access*, *10*, 12966–12982.

[6] Bhowmick, A., Sinha, N., Arjunan, R. V., & Kishore, B. (2017). Permutation-substitution architecture based image encryption algorithm using Middle Square and RC4 PRNG. *2017 International Conference on Inventive Systems and Control (ICISC)*.

[7] Ge, B., Chen, X., Chen, G., & Shen, Z. (2021). Secure and fast image encryption algorithm using hyper-chaos-based key generator and vector operation. *IEEE Access*, *9*, 137635–137654.

[8] Hamad, A., & Farhan, A. (2020). Image encryption algorithm based on substitution principle and shuffling scheme. *Engineering and Technology Journal*, *38*(3B), 98–103.

[9] Hua, Z., Zhou, Y., Pun, C.-M., & Chen, C. L. P. (2015). 2d Sine Logistic Modulation Map for Image Encryption. *Information Sciences*, *297*, 80–94.

[10] Li, T., Du, B., & Liang, X. (2020). Image encryption algorithm based on logistic and two-dimensional Lorenz. *IEEE Access*, *8*, 13792–13805.

[11] Lu, Q., Zhu, C., & Deng, X. (2020). An efficient image encryption scheme based on the LSS chaotic map and single S-box. *IEEE Access*, *8*, 25664–25678.

[12] Mazloom, S., & Eftekhari-Moghadam, A. M. (2009). Color image encryption based on coupled nonlinear chaotic map. *Chaos, Solitons & Fractals*, *42*(3), 1745–1754.

[13] Mishra, K., Saharan, R., & Rathor, B. (2017). A new cryptographic method for image encryption. *Journal of Intelligent & Fuzzy Systems*, *32*(4), 2885–2892.

[14] Panduranga, H. T., Naveen Kumar, S. K., & Kiran. (2014). Image encryption based on permutation-substitution using chaotic map and Latin square image cipher. *The European Physical Journal Special Topics*, *223*(8), 1663–1677.

[15] Pan, H., Lei, Y., & Jian, C. (2018). Research on digital image encryption algorithm based on double logistic chaotic map. *EURASIP Journal on Image and Video Processing*, *2018*(1).

[16] Samiullah, M., Aslam, W., Nazir, H., Lali, M. I., Shahzad, B., Mufti, M. R., & Afzal, H. (2020). An image encryption scheme based on DNA computing and multiple chaotic systems. *IEEE Access*, *8*, 25650–25663.

[17] Wang, X.-Y., Yang, L., Liu, R., & Kadir, A. (2010). A chaotic image encryption algorithm based on Perceptron model. *Nonlinear Dynamics*, *62*(3), 615–621.

[18] Wang, X.-Y., Zhang, Y.-Q., & Bao, X.-M. (2015). A colour image encryption scheme using permutation-substitution based on Chaos. *Entropy*, *17*(6), 3877–3897.

[19] Ye, G., Jiao, K., Huang, X., Goi, B.-M., & Yap, W.-S. (2020). An image encryption scheme based on public key cryptosystem and quantum logistic map. *Scientific Reports*, *10*(1).

[20] Zhang, R., & Xiao, D. (2020). A secure image permutation–substitution framework based on chaos and Compressive Sensing. *International Journal of Distributed Sensor Networks*, *16*(3), 155014772091294.

[21] The dataset of the human faces is accessed from the following link: https://www.kaggle.com/datasets/ashwingupta3012/human-faces.