A Project Report On "A Novel Image Encryption and Authentication Scheme using Chaotic Mapping and perceptual Hash"

Subject – Cyber Security

Course Code – CSE4003

Team Member:

Vrishab V Srivatsa - 18BCI0074

Under the guidance of Prof. Navamani T. M.

Table Of Contents:

Chapter	Title	Page No.
	Cover Page	**1**
1.	**ABSTRACT**	**3**
2.	**INTRODUCTION**	**3-5**
3.	**LITERATURE SURVEY AND PROBLEMS IN EXISTING SYSTEMS**	**5-9**
4.	**PROPOSED SYSTEM**	**9-20**
	OVERVIEW	
	SYSTEM ARCHITECTURE	
	FUNCTIONAL ARCHITECTURE	
	MODULAR DESIGN	
	INNOVATIVE IDEA	
5.	**IMPLEMENTATION**	**20-31**
	Software details and Screen shots with respective Description	
	Test cases and Analysis in terms of Table of Graph	
	Performance Analysis	
	Sample Code	
6.	**CONCLUSION AND FUTURE WORK**	**32**
7.	**REFERENCES**	**32**

Abstract

In this millennium age of 4G and 5G, the internet connectivity is faster and made available to billions across the globe at high speed and hence there is more and more flow of data. With the greater speed of the connections data size of data transfer is also increasing therefore encrypting only non-visual data is not alone making the cut. Security has acquired a lot of importance as information technology is used so widely in all sectors. Since, digital image has become one of the primary medium of communication, we have had to come up with different techniques from time to time to ensure confidentiality and authenticity of the images. Image encryption has proved important to protect media from any unauthorised breachs. Image and video encryption have applications in many different fields including social media and communication, multi-media systems, medical imaging, Tele-medicine and military communication. Hence, I have proposed a new and efficient way to encrypt visual data. The problem statement I have taken up was on distorting an image with the help of a secret key and reconstructing it again in an optimal manner, and at the same time verifying the authenticity of the image.

Although many have a proposed a way to encrypt the images, most of them have used grayscale images to do so, while others are mostly based on hiding text in images (steganography).

This project proposes a new algorithm to encrypt and decrypt images while comparing to other modern encryption approaches and ultimately uncovers a conclusion and suggests future work.

Introduction

The Internet is rapidly flourishing. As a result, people use digital media extensively in communication. For example, image, audio, and video. Images occupy a large percentage of the digital media. Images play a key role in communication, for example, army, national defence, strategic affairs, personal images, images of private documents. Because these images contain potentially sensitive information, these images provide extreme protection when users accumulate over an untrustworthy database. Furthermore, when people want to transfer images over an unsafe network, it becomes essential to provide complete security. In short, an image requires protection against a variety of factors to be able to provide privacy to the users. Keeping pictures shielded is primarily intended to preserve confidentiality, honesty and validity. Various methods are accessible to safeguard pictures, and encryption is one method. The adoption and application of different encryption algorithms can guarantee data security from spoofing and eavesdropping from unlawful assaults and security-analyst.

Image encryption has been done by various methods depending on various factors its efficiency is defined by various factors like key sensitivity, histogram, coefficient of correlation and key space if there. The main problem that arises that is hard as compared to the that of word message to rearrange/destruct the data and then able to construct the data back. As the data suggests that methods like DES, IDEA, AES and blowfish are not appropriate for encryption especially if it is over an unsafe network, if used the for encryption of an image these do not alter the unique characteristics of images like data redundancy and no predetermined distribution of pixels demanding high computation power and time. Plus, the key to encryption has to be lengthy and it is comparatively hard to remember it, however, and it is even insecure to store the key in a database or file. Additionally, protecting encryption keys ' confidentiality is one of the significant problems to be addressed. This problem can be solved effectively by creating the key before beginning the encryption / decryption process, instead of storing it.

In recent times there have been many different methods proposed like transformation and XOR, three one-dimensional chaotic map, bit level permutation, XOR & rotation, two-dimensional logistic map & compression and multi-dimension chaotic function but every one of them has different issues in them some have low number NPCR (number of pixel change rate) or high coefficient of correlation (the correlation between two adjoining pixels of the original and the encrypted image) or have low key sensitivity (generating completely contrary consequence despite a small change in key) or small key size, as large key space will help to defend against brute forcing of the key making the total combination too large, to the encryption process all of which is essential in encrypting the image. Achieving all of them simultaneously had been a challenge for most of the known algorithm.

Hence there is a need to devise to better and efficient way of making the transfer of visual messages more secure and can be used in different applications over an untrusted network. Hence by assuming the image as a matrix of dimensions (length, width) and using the defined vast properties of matrix we are first XORing the pixel RGB values with the transformed secret key.

The transformation of matrix along one of the sides of the matrix could completely shuffle the arrangement of the image making the image unrecognisable, making the image more secure, even better if there are more pixels in the image.

Organization of the project is completely based on Python and a few dependencies, while taking the test results as random images from various websites. Depending on the various functionalities we can easily calculate the various parameters of the encryption like difference in histograms between the original and encrypted image that will tell us how much the intensities of the RGB components of the encrypted image is different from the original image and would be very hard to brought back from the encrypted and would take a lot of time and computational power to do so.

Related Work/Literature Survey

Image Encryption Using Affine Transform and XOR Operation

Image encryption is an appropriate method to protect and secure image data. Image data and text data have their own features unique to them. Present encryption algorithms available are suitable for text data, but they are not really suitable for multimedia data such as images. In actuality, the correlation to their neighboring pixels of natural photos are very high. Due to the high correlation values, any pixel of the image can be logically predicted from the values of its neighboring pixels. In this article, a new location-transformation based encryption technique has been proposed. The pixel values are redistributed to different locations using a technique called affine transform, which uses four 8-bit keys. The transformed image is then divided into blocks of sizes 2 x 2 pixels and each block is then encrypted. The encryption technique uses XOR operation by another four 8-bit keys. The key size used in this algorithm sums up to 64 bit which proves to be pretty strong. Upon experimentation and noting down the results, it could be proved that after the affine transform the correlation between pixel values was significantly decreased.

In the past there has been many proposed methods like transformation and XOR, RC4 cipher, DES, 3-DES, AES, ECC, Chaotic map etc. while some of them relate to classical ciphers, most of them comes under the category of modern ciphers.

Affine Transformation and XOR

Encryption:

A 256 grey level secret image of size MXN and 64 bits secret key is taken as input.

In the first step the 64 bit key is broken into 8 equal parts namely K_0 K_1 K_2 K_3 K_4….K_7.

For each pixel transforming the location coordinates using $x^{'}=(K_0+K_1*x) \bmod M$ and the y coordinate $y^{'}=(K_2+K_3*y) \bmod N$.

Decomposing into (M*N)/4 number of 2X2 blocks.

For each block B_{ij} of cipher image

$P'_{1.1}=P_{1.1}\oplus K_4$

$P'_{1.2}=P_{1.2}\oplus K_5$

$P'_{2.1}=P_{2.1}\oplus K_6$

$P'_{2.2}=P_{2.2}\oplus K_7$

The image is encrypted.

Decryption algorithm

Decryption happens when the cipher image is decomposed into MXN/4 number of 2X2blocks. Each pixel of every block is then XOR-ed with 4 least significant bits. The decrypted pixels are then restored back to their original position using the given equations.

$X=(x'+(-K_0))XK_1^{-1}\text{mod } M$

$Y=(y'+(-K_2))XK_3^{-1}\text{mod } N$

Elliptic curve cryptography

Elliptic curve cryptography, based on elliptic curves, as shown in Fig.1, is the public key device. Neal Koblitz and Victor Miller were the ones who individually put forward the technique in 1985. It provides many facilities in over other public cryptosystems one main of which is that it provides a smaller, faster public key cryptosystem. A 160-bit key in ECC is considered as safe as 1024-bit key in RSA. It is based on the Elliptic Curve Discrete Logarithm issue, recognized as NP-Hard problem. The definition of an elliptic curve is revealed by the equation, $Y^2 = x^3 + ax + b$. In the equation, a finite field is represented by a , b and P_n elements, P is a prime larger than 3, and $4a^3 + 27b^2 \neq 0$. The overall points on the curve are the group of ordered pairs (x, y) and selected items in the field and such as x and finite field ($P \neq 2$ or 3), two affine points and distinct x-coordinates are added, and the result of addition or subtraction in the field is ignored.

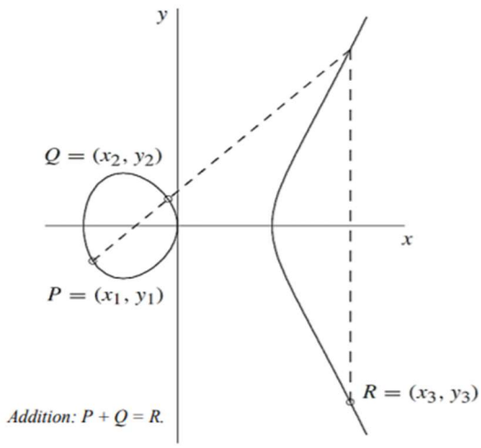


Fig. 1 Elliptic Curve Cryptography

P is chosen on the curve and is used as an element of public key while any random of integer K is taken as the private key. K is a scalar and Q is also point on an elliptic curve can be calculated Q = K*P, where P, Q ∈ E(K) and P+Q ∈ E(K). Finally, the sum of points P and Q is an equal to the R is a reflection in the x-axis of the third point at which this line intersects the curve. E, P, n and Q together makes the entire public key.

Table 1 Literature Survey

JOURNAL	METHOD USED	ADVANTAGE	DISADVANTAGE
A secure and robust hash-based scheme for image authentication	Randomized pixel modulation Hash collision Wavelet transform	Robustness Tamper detection	High possibility of a false negative in tamper detection
Image Authentication using Image Hashing	Feature extraction, Ring partition, Vector distance, Image hashing.	Increased Hash Distance Robust against content preserving operations	Forged area localization Can include the local features to enhance the hash

based on Ring Partition with Corner Inclusion			performance so as to reduce the error probability.
Image Encryption Using Affine Transform and XOR Operation	Affine transform, Symmetric key encryption.	Correlation between pixel values was significantly decreased.	Key size fixed to 64 bits
Image Encryption with Double Spiral Scans and Chaotic Maps	Random overlapping block partition, double spiral scans, Henon chaotic map, and Lü chaotic map	Choosing different start-points for different image blocks by using Henon chaotic map. This strategy can improve security of this algorithm.	Lossy compression ie image might not retain 100% of it's features
Image Encryption and Decryption in Public Key Cryptography based on MR	Magic Rectangle Encryption	It is observed that using the compression technique, the file size is reduced as approximately half of the original size. It helps to reduce the processing time.	Reduce the additional time needed for construction of magic rectangle and compression of images.

Identified Problems in Existing work

Avoid lossy compression

Some algorithms insisted that the decrypted image had been compressed and key data was lost in the process.

Reduce time complexity and computation time

In real world applications, and in bulk encryption, time is a key factor to acknowledge while designing a new algorithm.

Must be robust

Security cannot be compromised. Image correlation and other factors must support the proposed method.

Must work for grayscale as well as colored images

Must work for all formats of images

JPEG, PNG, SVG, etc to name a few

Key size should be flexible

Algorithms like AES, Blowfish have fixed key sizes, where padding is used. In proposed method, key size is flexible as length of key array (MD5 hash digest) is always fixed, and does not affect the performance.

Proposed System

Overview

The proposed image encryption/authentication technique mainly involves the concepts of XOR of RGB components of the pixels in the image, in a cipher block chaining mode followed by elementary row and column transformations.

The initialisation vector here is nothing but the transformed secret key, which itself is an array consisting of the ASCII values of the hexadecimal hash digest of the entered key. For instance, let's consider the secret key: "Password123"

MD5 hash digest: 42f749ade7f9e195bf475f37a44cafcb

Array of hash digest: [4,2,f,7,4,9,a,d,e,7,f,9,e,1,9,5,b,f,4,7,5,f,3,7,a,4,4,c,a,f,c,b]

Array of ASCII values (IV): [52, 50, 102, 55, 52, 57, 97, 100, 101, 55, 102, 57, 101, 49, 57, 53, 98, 102, 52, 55, 53, 102, 51, 55, 97, 52, 52, 99, 97, 102, 99, 98]

Once the above calculations are done, it is proceeded by a cipher block chaining XOR encryption of the image pixels, iteratively. Each loaded pixel has three color

components: Red, Green and Blue. The XOR encryption of pixel [q,r] is done as follows:

reds=pix[q,r][0] xor (key_array[q*r%key_length]**2%255) xor (key_length*key_sum%255)

greens=pix[q,r][1] xor (key_array[q*r%key_length]**2%255) xor (key_length*key_sum%255)

blues=pix[q,r][2] xor (key_array[q*r%key_length]**2%255) xor (key_length*key_sum%255)

pix[q,r] = (reds,greens,blues)

This is followed by an iterative pixel swapping. The elementary row transformation, as shown in Fig. 2, essentially used recursively in the algorithm is the interchanging of one row of the matrix completely with another row of the matrix, over and over repeatedly.

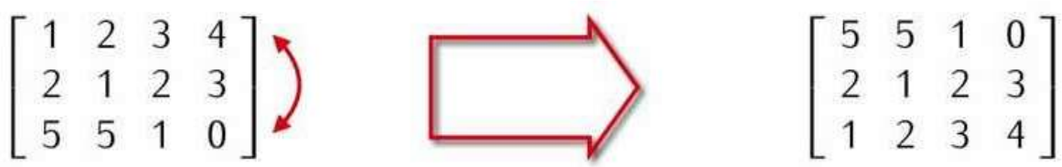


Fig. 2 Elementary Row Transformation

Similarly, we have used column transformations to interchange given two columns of the matrix. When paired together and repeated, row and column interchanges give a completely different matrix compared to the initial state.

This concept has been linked with images here. An image can be visualised as a matrix with m rows and n columns, where every pixel can be compared to a matrix element, where m is the height of the image and n is the width of the image in pixels. The interchanging of rows and columns is followed in quite a complicated pattern, and is totally dependent on the key entered by the user. No two keys are expected to result to the same encrypted/decrypted image and hence the sensitivity of a key is levelled up in a certain way.

We start with reading the given image. The height and width of the image is stored as variables in the program. The user then provides a secret encryption key which he/she expects only the receiver of the encrypted media will know. The key length can be varied. It is suggested to use a longer key (128+ bits) as it is harder to be retraced.

However, our algorithm will work regardless of the length of the key provided. Then, we begin the encryption phase.

In this encryption phase, we interchange every i^{th} row of the image with (i+n)modulo(x)th row, where n corresponds to the (k)modulo(length of key)th ASCII value of the key, where k is another variable running in a loop. The same is done with every i^{th} column as well. This whole process of interchanging rows and columns is repeated a number of times, which is directly proportional to the size of the image. This distorts the image and makes it completely incomprehensible. Any attacker that may access the encrypted image will not be able to identify the image with ease, due to the distortion of original image. He/She will need the complete key in order to decrypt the image that has been encrypted using our algorithm. Even a wrong key that differs by 1 bit from the original key will not lead to successful decryption. Hence, key sensitivity can be attributed as one of the leading advantages of our algorithm. The understanding of the gist of our encryption/decryption process can be made clearer with the Architecture Diagrams.

System Architecture

System components, are shown below in Fig. 3

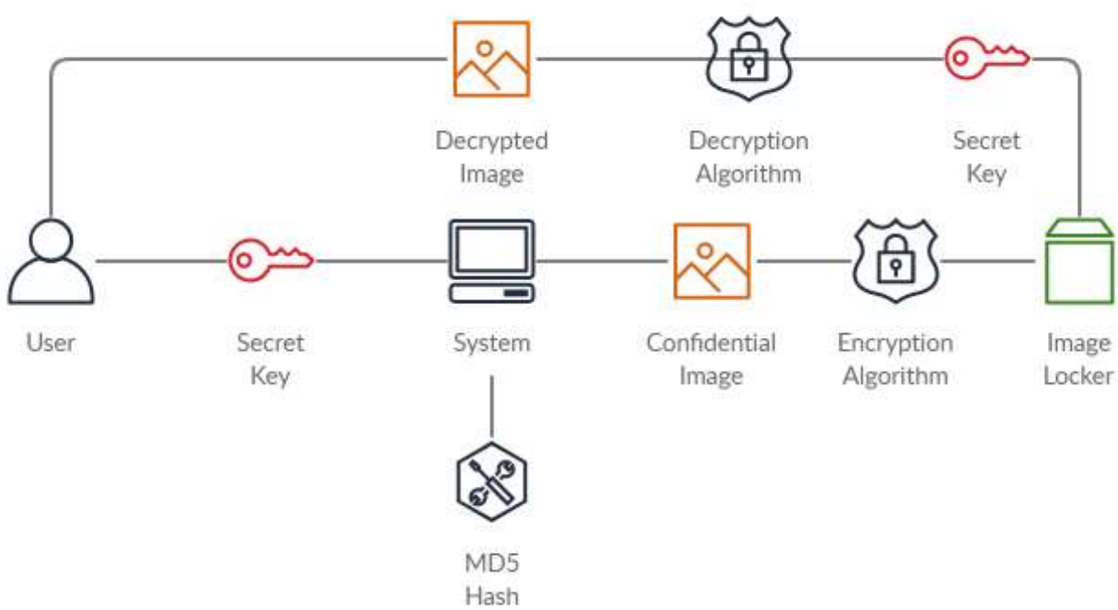


Fig. 3 System Architecture Diagram

Functional Architecture

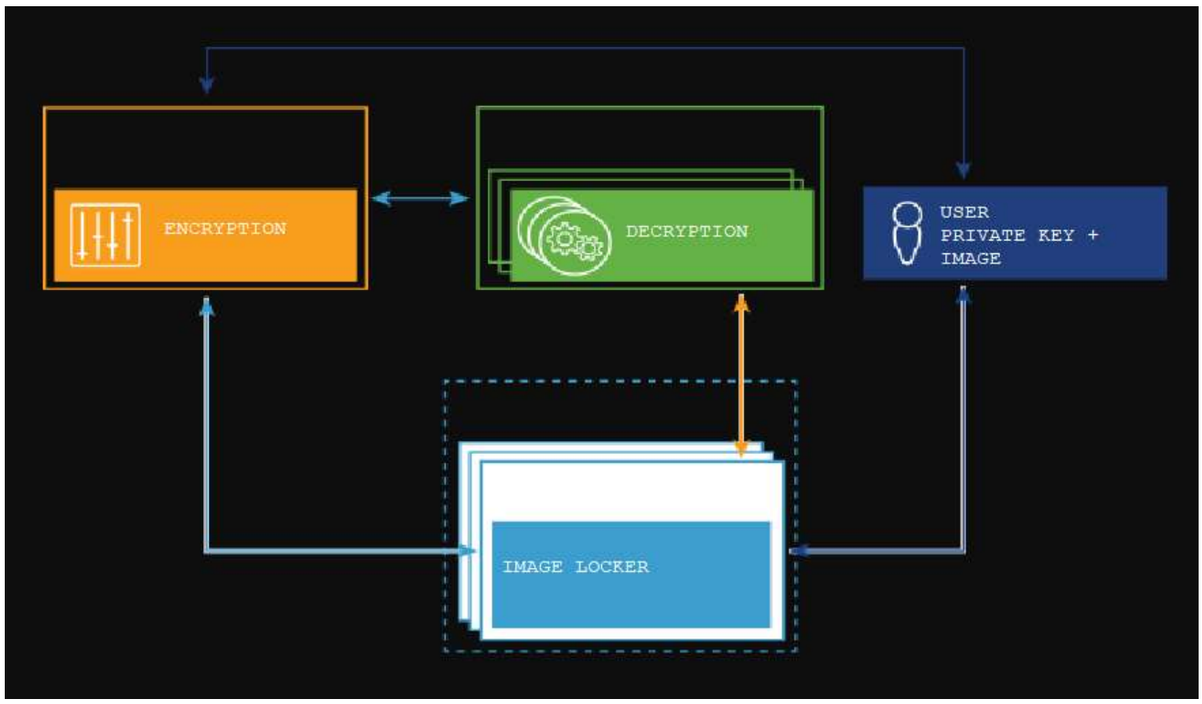


Fig. 4 Functional Architecture

Modular design

i) Key hashing and IV

The secret key entered by the user is hashed, using MD5 algorithm, to obtain a hash digest. This hexadecimal has is converted to an array, and each element of the array is stored as it's ASCII value. This array serves as the key array or initial vector in the encryption process, as shown below in Fig. 5

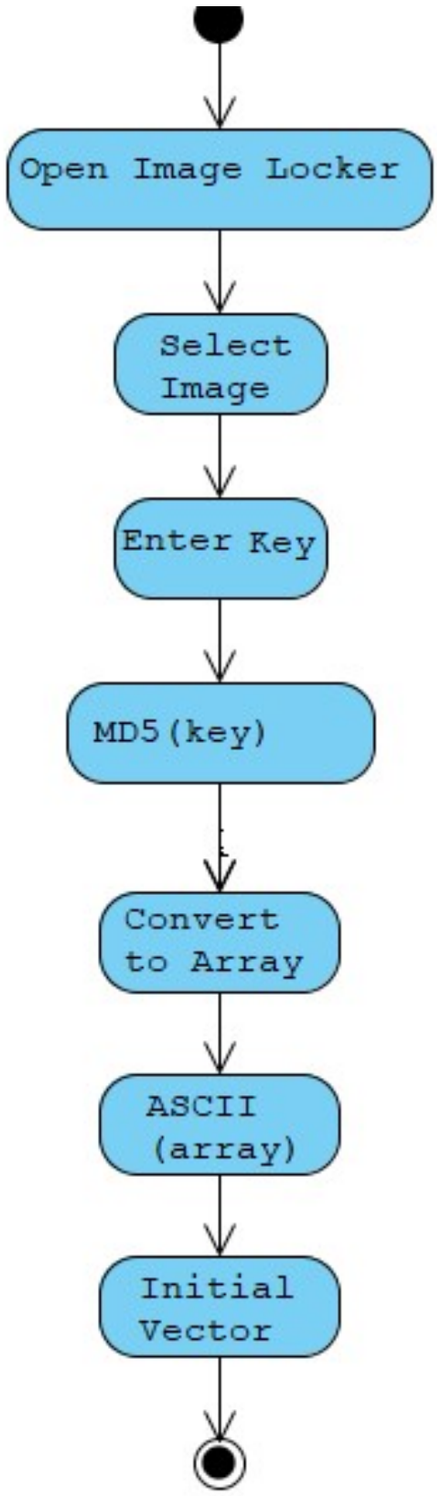


Fig. 5 Key Hashing Process Flow

ii) Image Hashing

p-Hash

Perceptual hash algorithms describe a class of comparable hash functions. Features in the image are used to generate a distinct (but not unique) fingerprint, and these fingerprints are comparable any ways, as shown below in Fig 6.

A distance of 5 means a few things may be different, but they are probably still close enough to be similar. But a distance of 10 or more? That's probably a very different picture.

Along with p-Hash primarily, we have also calculated average-hash, colorhash, wavelet hash and distance hash of the original and decrypted images, to ensure that there is no tampering or alteration in bits of the pixel RGB intensity values either by an intruder or network loss.

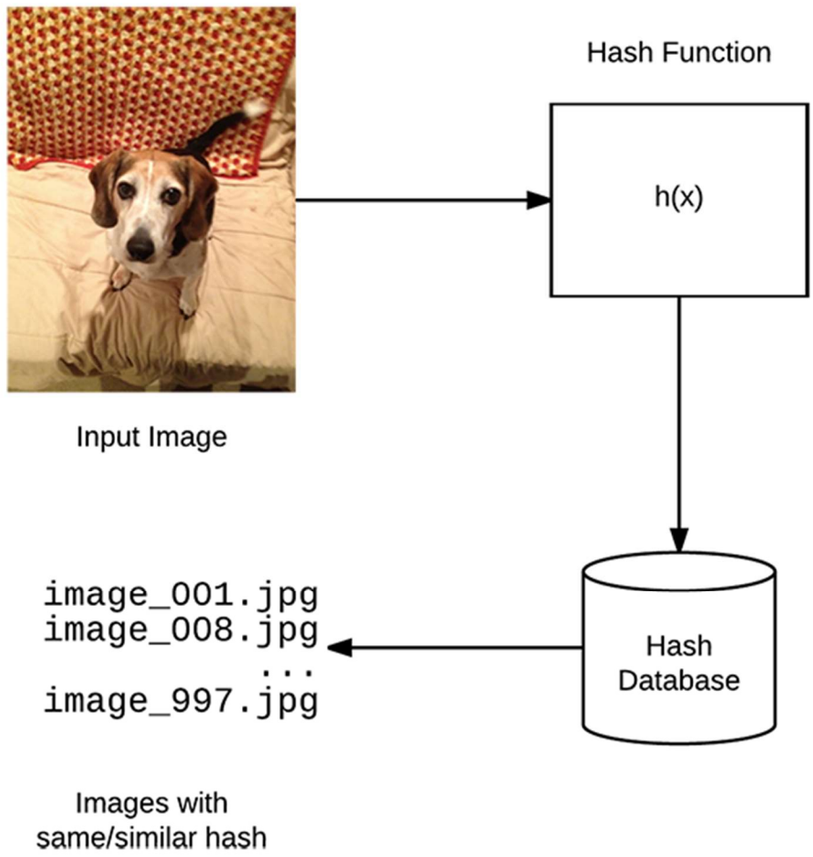


Fig. 6 Image Hash Functions

iii) XOR Chain Block Cipher Encryption

The RGB components of the pixels of the private image are XORed one by one iteratively, while depending on the preceding pixel for its result, as show in Fig. 7. The color components are also XORed by the sum of the key array to add more weightage to key sensitivity.

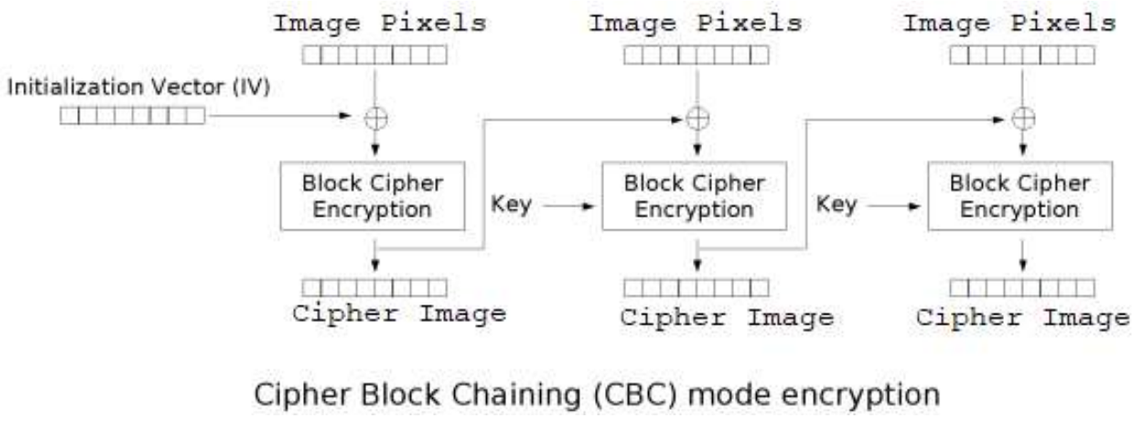


Fig. 7 CBC Encryption

iv) Pixel Shuffling

Algorithm:

An encryption algorithm is as secure as it's key, so we had to insert the key in a very careful way, while preserving both key sensitivity and distorting the image well enough to not be identifiable. We chose to create a function f(x) which swapped every j^{th} row (as explained above) and every i^{th} column with $k(x)^{th}$ column as defined below, running in the loop. Loopholes were identified and corrected to obtain a reliable function f(x).
We define f(x) as,

$$f(x): \begin{cases} j \rightarrow (j + k * (key[k\%(length(key)]) * key\_length^1)\ \%n \\ i \rightarrow (i + k * (key[k\%(length(key)]) * key\_length^1)\%m \end{cases}$$

where i, j, k are the loop variables, n and m are the width and height of the image, key[in] is the index of the element of key array and key is the secret key given by the user.

Pixel shuffling Flow Diagram:

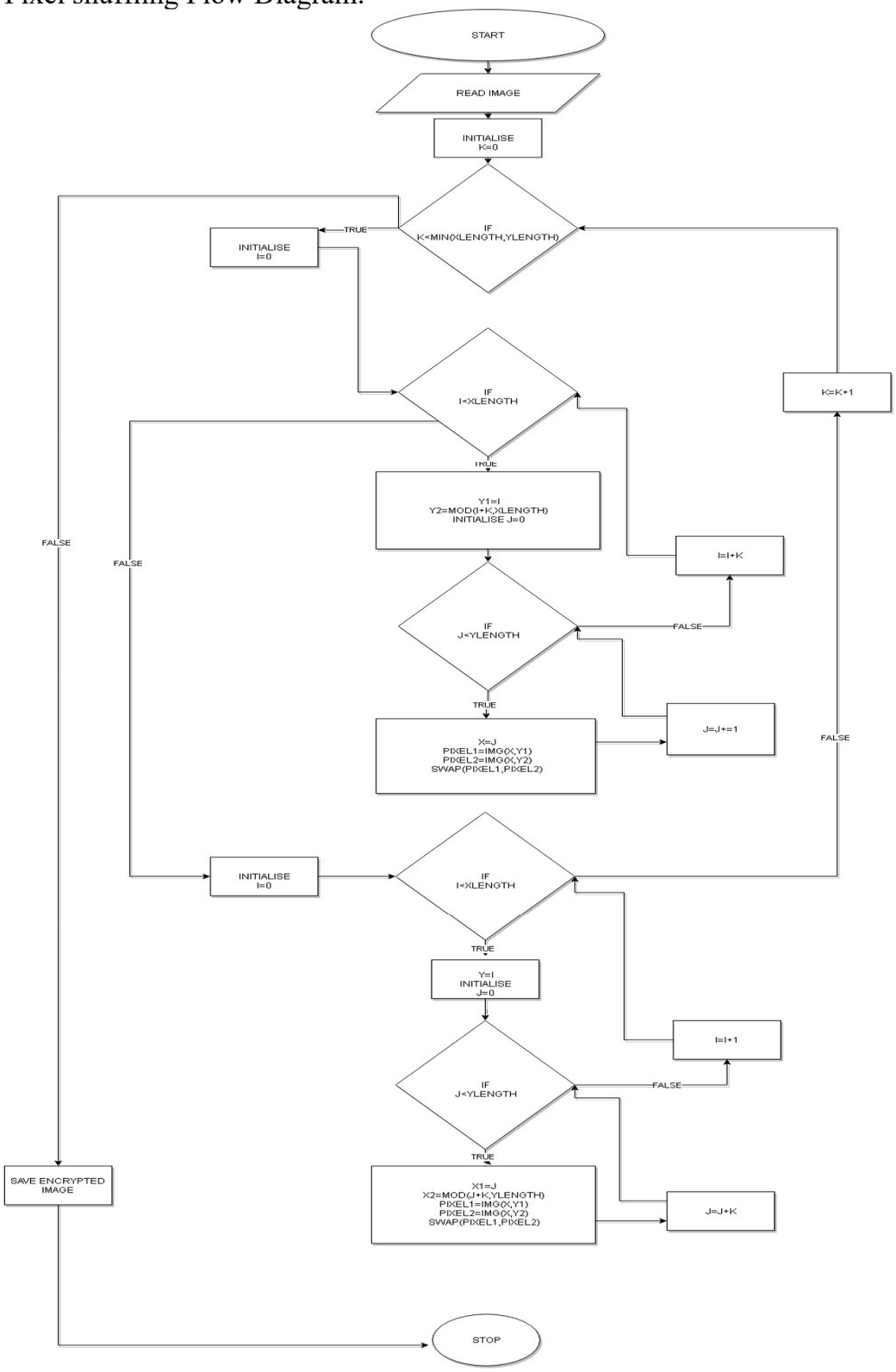


Fig.8 Pixel Shuffling

v) Pixel Rearrangement

The decryption phase is basically reversing the encryption algorithm, with the help of secret key. It takes same computational time as encryption. Without the exact same key, we cannot get back our original image by any ways.

The same f(x) is used to recalculate original position of each pixel, and the loop is iterated in reverse this time.

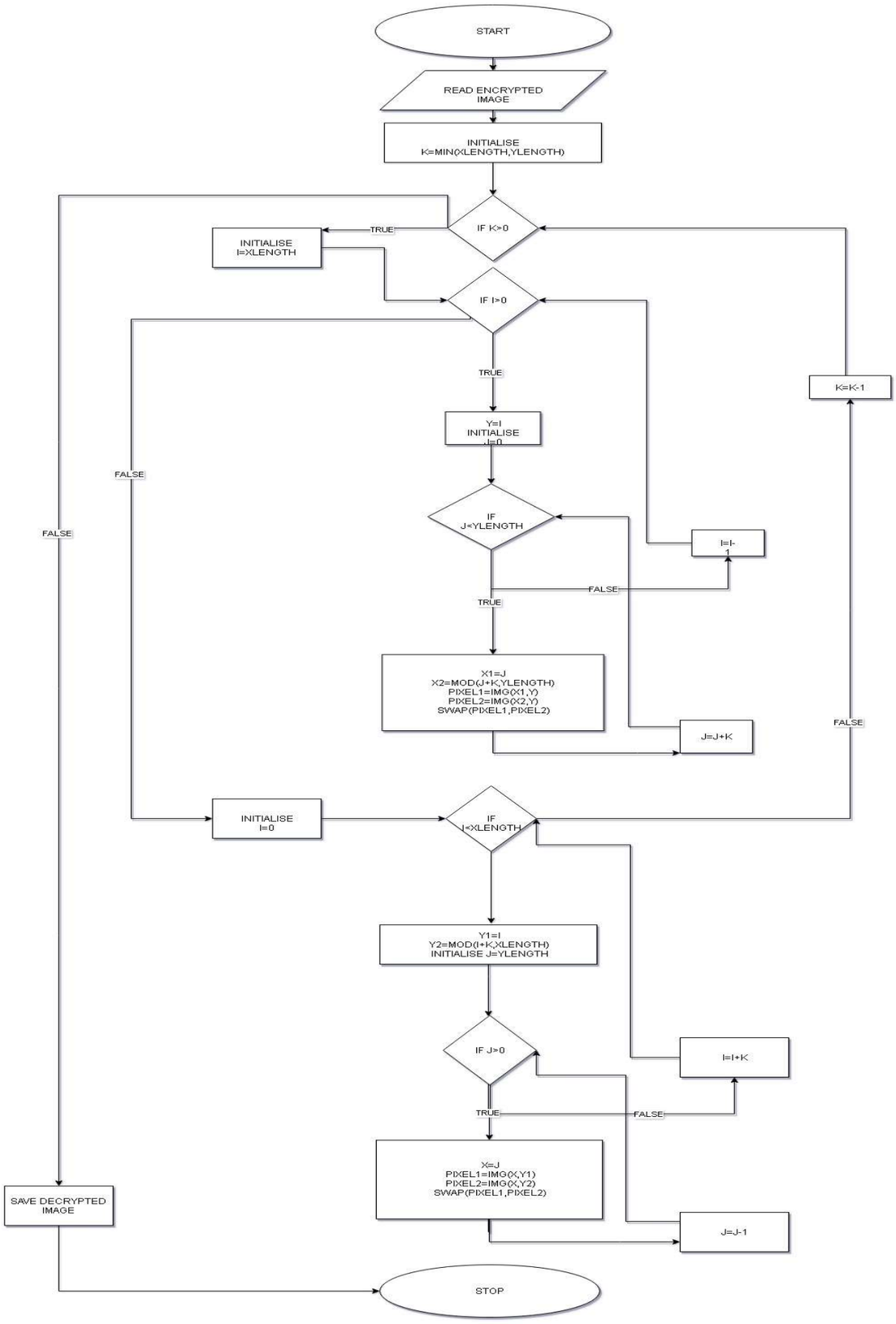


Fig. 9 Pixel Rearrangement

vi) XOR Chain Block Cipher Decryption

The exact same process done during encryption is repeated once again.
The RGB components of the pixels of the private image are XORed one by one iteratively, while depending on the preceding pixel for its result. The color components are also XORed by the sum of the key array to add more weightage to key sensitivity.

We know that

A xor B xor B = A xor (B xor B) = A xor (0) = A

Therefore, upon CBC Decryption, we get back the original image.

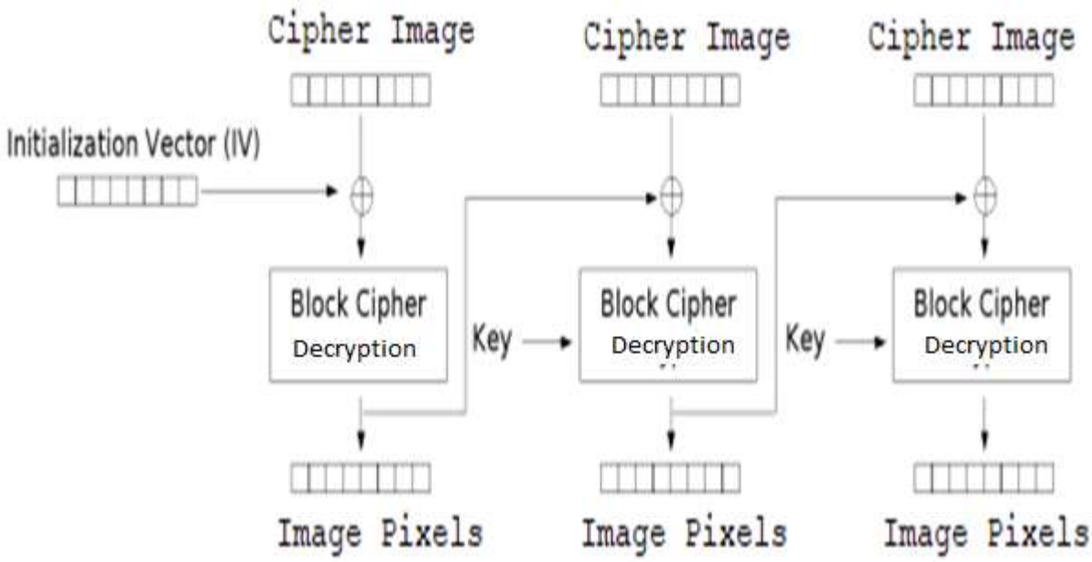


Fig. 10 CBC Decryption

vii) Image Hashing

This time, the hashing of image is done to compare with the original image, to ensure that there is no loss of data or any forms of tampering.

Innovative idea in project

1. The algorithm is original, ie, never been proposed before, while also competing out with a balanced performance and comparability with existing algorithms.

2. Based on a simple concept of shuffling a Rubik's cube, and solving it
3. An added option to customise the levels of encryption, which will help users prioritise performance and security accordingly to their needs.
4. Can work well with Military communication just as well for a simple naive personal use.
5. Addition of image hashing ensures authenticity of image, to ensure that there is no alteration in the information stored in the image, making it even more secure.
6. Image locker, our application that can help users encrypt images in their PC, for no cost compared to softwares which will do the same job while charging you a lot of money!

Implementation Details and Analysis

We have implemented the whole project using Python 3.8.
The project has been tested extensively with many test cases, bounded inputs to recognise faults and corrections were made accordingly.

The required Open Source modules on Python are:

cycler==0.10.0
ImageHash==4.1.0
importlib-metadata==2.0.0
matplotlib==3.3.2
numpy==1.19.2
opencv-python==4.4.0.44
Pillow==8.0.1
pyparsing==2.4.7
PySimpleGUI==4.30.0
python-dateutil==2.8.1
PyWavelets==1.1.1
scipy==1.5.3
soupsieve==2.0.1
sqlparse==0.4.1
zipp==3.4.0

Test cases / Analysis with Graphs

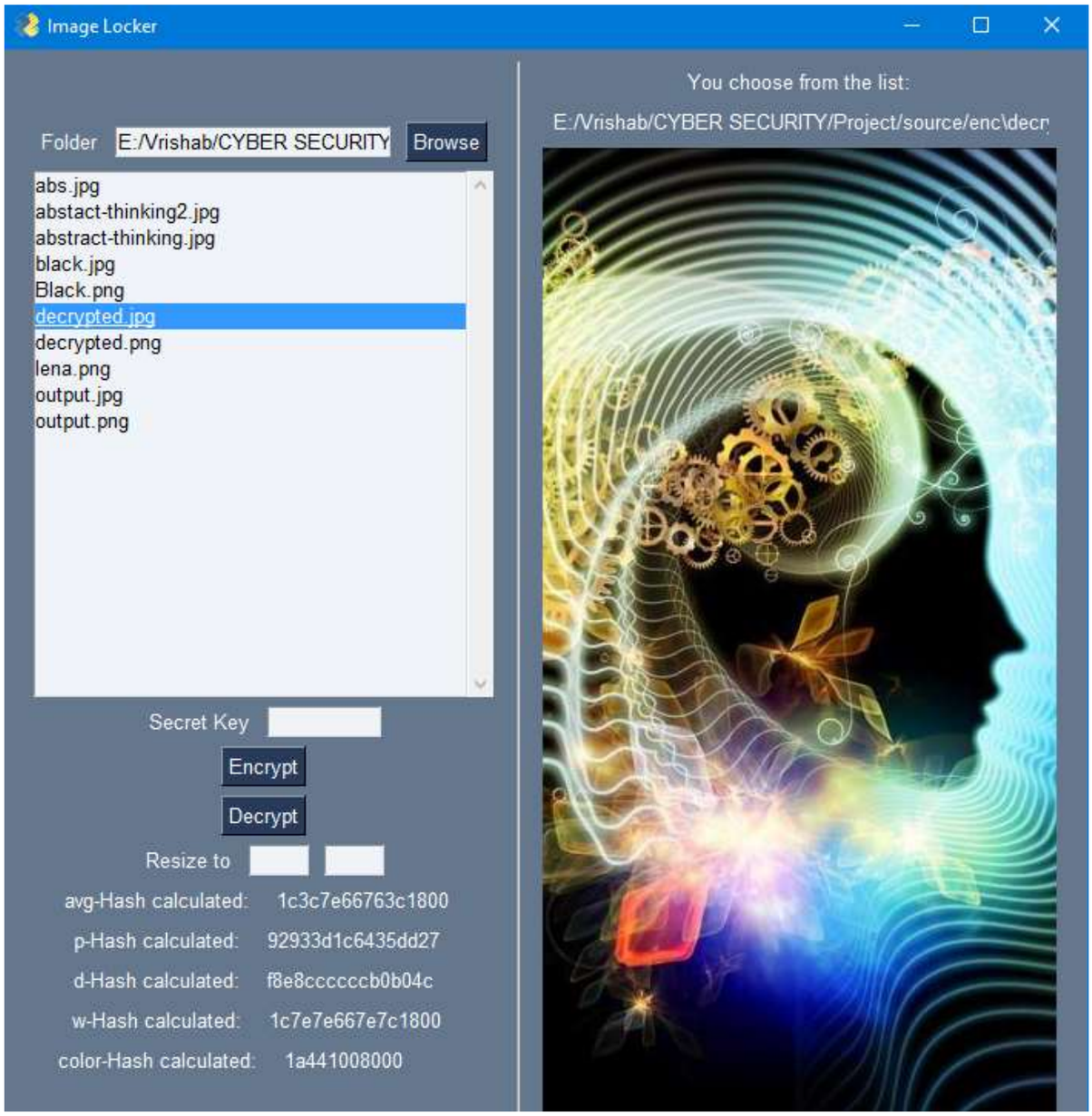


Fig. 11 The designed GUI

1) The famous Lena Portrait

Fig. 12 Original Image

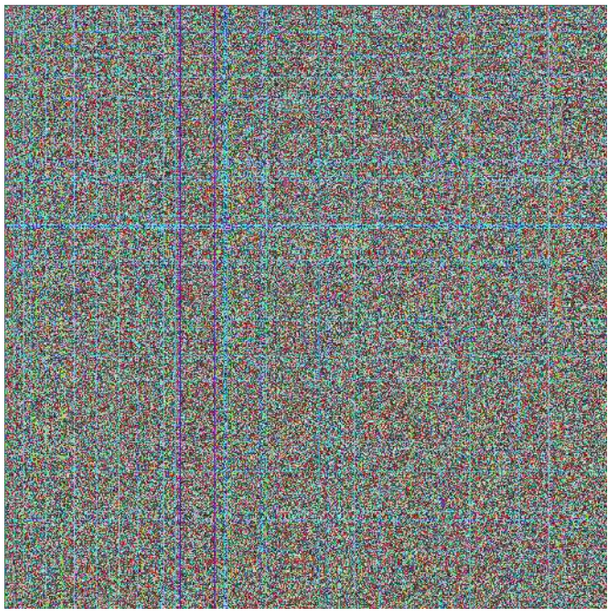


Fig. 13 After Encryption with the key : "CyberSecurity"

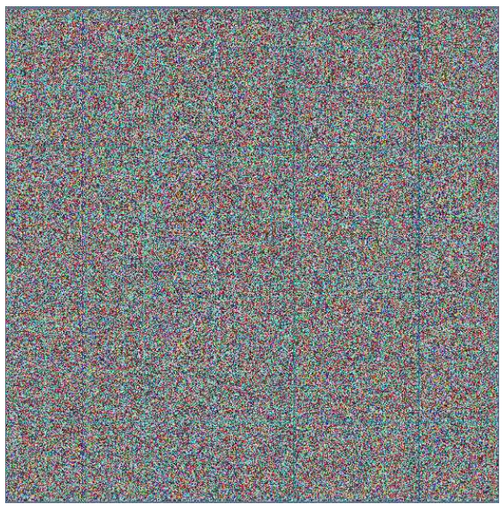


Fig. 14 Decrypting with the wrong key

Fig. 15 Decrypting with correct key

2) A plain black image

Fig. 16 Original Image

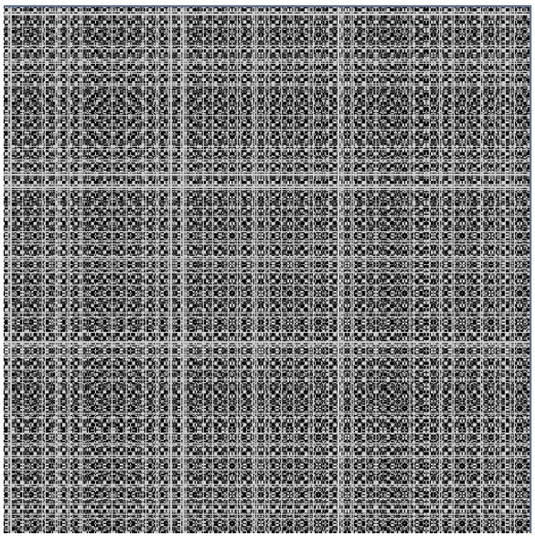


Fig. 17 Encrypting with the key "Black123"

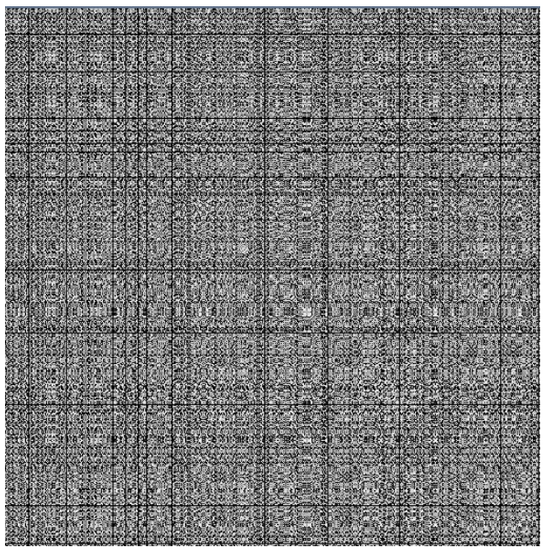


Fig. 18 Decryption with a wrong key

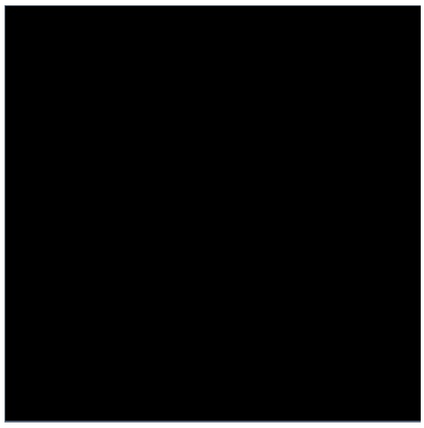


Fig. 19 Decryption with correct key

Performance Analysis

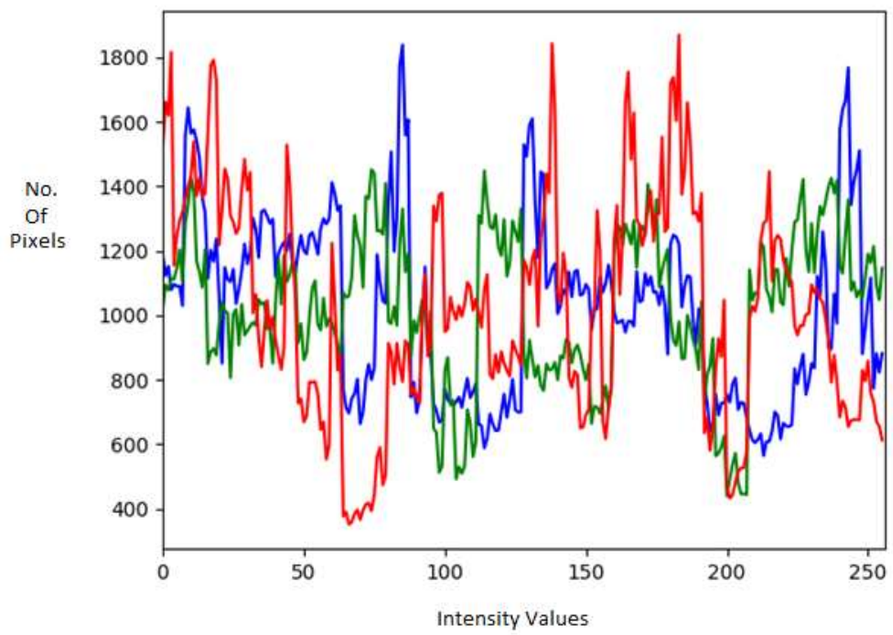


Fig. 20 Histogram of encrypted – Lena

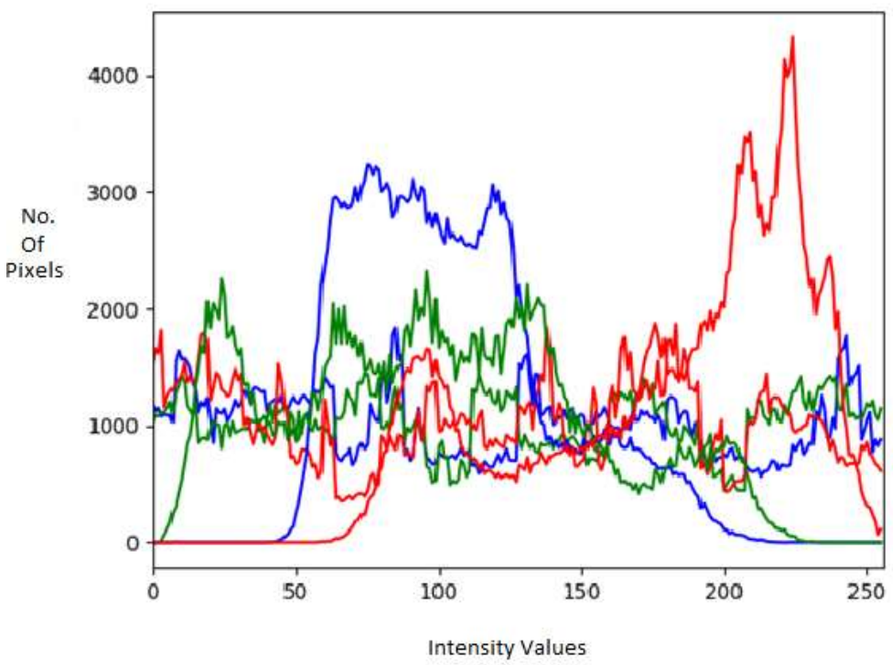
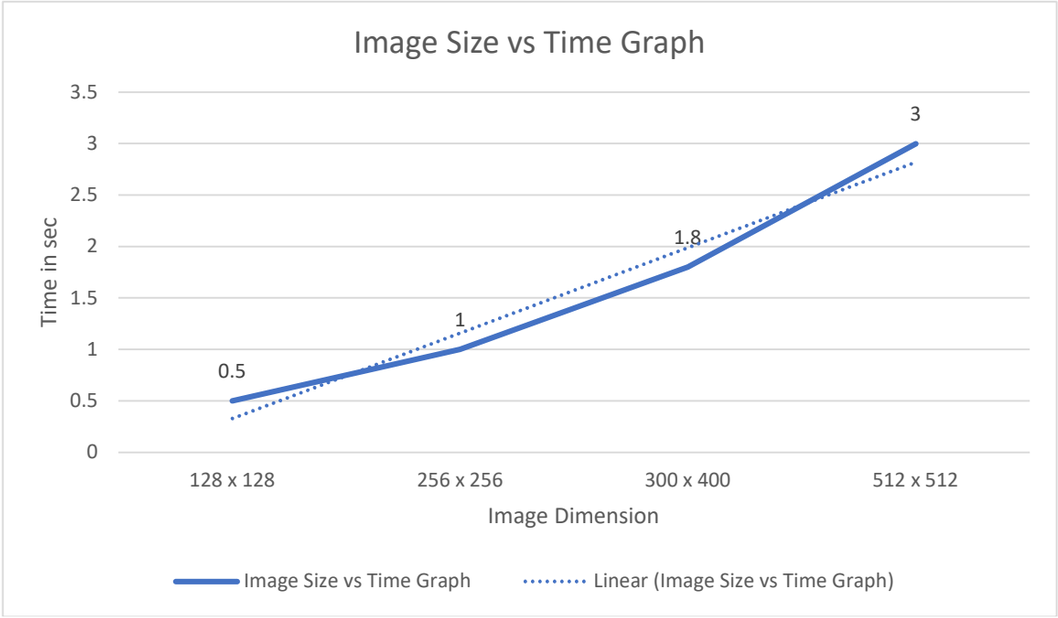


Fig. 21 Histogram of original image – Lena

Time Complexity

Figure. 22 Time Complexity Graph



From the graph, time complexity was found to be O(M x N), where M and N are the height and width of the image.

DISCUSSION:

A thorough scrutiny of the techniques is done using diverse parameters that have been put into assessment.

Visual Assessment
In any efficient encryption method, the encrypted images must not contain any visual information or resemblance to the original image. They should be highly distorted so that it will impossible for any intruder to crack even a portion of the image.

Key-Length and Key-Space Analysis
A basic feature of image encryption algorithms is sensitivity to keys, initial value of parameters used and length of keys. Reduction of brute force attacks is due to large key space. In our algorithm, the key length can be varied, as per the user's choice. Large key length (above 128 bit) ensures maximum security and protection from Brute Force Attacks.

Correlation Analysis
This parameter is used to calculate the correlation coefficient among the adjacent pixels of an image. A good encryption technique should result into an encrypted image with no or very little correlation (<1%) between nearby pixels. A correlation between pairs of plain and encrypted image channels has been calculated. Our algorithm shows very little correlation (0.05%-0.1%) for any given image.

Pixel/Plain image sensitivity
Pixel sensitivity means that an entirely different and unique encrypted image is the obtained when even tiny change is made in the plain image. The two encrypted images can be differentiated by calculating NPCR. We obtain at least 99% NPCR for any two given encrypted images for the same plain image, which is an excellent figure.

Key Sensitivity
Key sensitivity is used to analyse the effect of decryption when a minor change is made in the key during decryption. We changed the key by a minimum of 1 bit from the original key, and found no resemblance between the decrypted image and original plain image.

From repeated testing and analysis, we have obtained the following results about the performance of our image encryption algorithm, as compared to other algorithms.

Method used	**Visual Scrambling**	**Correlation Co-efficient Value**	**Pixel Sensitivity**	**Key Space**	**Key Sensitivity**	**NPCR**
Proposed Method	High	Low	High	128bit	Very High	High
Transformation and XOR [6]	Good	Medium	High	Large	Low	High
ECC Image Encryption [7]	High	Low	High	Large	High	High

Table. 2 Comparison of Performance

Sample Code - Python 3.8

Encryption

```
from PIL import Image
import hashlib
import cv2
from matplotlib import pyplot as plt

def encrypt(image,key):
    im = Image.open(image)
    pix = im.load()
    size = im.size
    mod = min(size)
    enc_key = key
    key = hashlib.md5(key.encode()).hexdigest()
    print(key)
    key_length = len(enc_key)
    key_array = []
    key_sum = sum(key_array)

    for key in key:
        key_array.append(ord(key)%mod)
    print(key_array)
    for q in range(size[0]):
        for r in range(size[1]):

reds=pix[q,r][0]^(key_array[q*r%key_length]**2%255)^(key_length*key_sum%255)

greens=pix[q,r][1]^(key_array[q*r%key_length]**2%255)^(key_length*key_sum%25
5)

blues=pix[q,r][2]^(key_array[q*r%key_length]**2%255)^(key_length*key_sum%255
)
            pix[q,r] = (reds,greens,blues)
    for k in range (2,key_length):
        for i in range (0,size[0],k):
            y1=i
            y2=(i+k*(key_array[k%(key_length)])**key_length)%size[0]
            for j in range(0,size[1]):
                x=j
                pixel1 = pix[y1,x]
                pixel2 = pix[y2,x]
                pix[y1,x] = pixel2
                pix[y2,x] = pixel1

        for i in range (0,size[0]):
```

```
            y=i
            for j in range(0,size[1],k):
                x1=j
                x2=(j+k*(key_array[k%(key_length)])**key_length)%size[1]
                pixel1 = pix[y,x1]
                pixel2 = pix[y,x2]
                pix[y,x1] = pixel2
                pix[y,x2] = pixel1
    im.save(image)
    img = cv2.imread('decrypted.png')
    color = ('b','g','r')


    for i,col in enumerate(color):
        histr = cv2.calcHist([img],[i],None,[256],[0,256])
        plt.plot(histr,color = col)
        plt.xlim([0,256])
    plt.show()
```

Decryption

```
from PIL import Image
import hashlib
import cv2
from matplotlib import pyplot as plt

def decrypt(image,key):
    im = Image.open(image) # Can be many different formats.
    pix = im.load()
    size = im.size
    mod = min(size)
    enc_key = key
    key = hashlib.md5(key.encode()).hexdigest()
    print(key)
    key_length = len(enc_key)
    key_array = []
    key_sum = sum(key_array)

    for key in key:
        key_array.append(ord(key)%mod)
    print(key_array)
```

```
    for k in range (key_length-1,1,-1):
        for i in range (size[0]-1,-1,-1):
            y=i
            for j in range(size[1]-1-(size[1]-1)%k,-1,-k):
                x1=j
                x2=(j+k*(key_array[k%(key_length)])**key_length)%size[1]
                pixel1 = pix[y,x1]
                pixel2 = pix[y,x2]
                pix[y,x1] = pixel2
                pix[y,x2] = pixel1

        for i in range (size[0]-1-(size[0]-1)%k,-1,-k):
            y1=i
            y2=(i+k*(key_array[k%(key_length)])**key_length)%size[0]
            for j in range(size[1]-1,-1,-1):
                x=j
                pixel1 = pix[y1,x]
                pixel2 = pix[y2,x]
                pix[y1,x] = pixel2
                pix[y2,x] = pixel1

    for q in range(size[0]):
        for r in range(size[1]):

reds=pix[q,r][0]^(key_array[q*r%key_length]**2%255)^(key_sum*key_sum%255)

greens=pix[q,r][1]^(key_array[q*r%key_length]**2%255)^(key_sum*key_sum%255)

blues=pix[q,r][2]^(key_array[q*r%key_length]**2%255)^(key_sum*key_sum%255)
            pix[q,r] = (reds,greens,blues)
    im.save(image)
    img = cv2.imread(image)
    color = ('b','g','r')


    for i,col in enumerate(color):
        histr = cv2.calcHist([img],[i],None,[256],[0,256])
        plt.plot(histr,color = col)
        plt.xlim([0,256])
    plt.show()
```

Conclusion and Future Work

Image Encryption algorithms are useful in various different applications such as secure messaging, sharing images on social media, in military/ defence sector, Watermarking and Steganography techniques, medical image encryption, etc. Hence, algorithms such as ours play a vital role in maintaining security and authenticity. Image encryption algorithms will never run out of scope in the future, and further improvements in existing algorithms and proposals of new algorithms will be highly appreciated by the community. It is safe to conclude that our proposed method is reliable and performs as well compared to most of the discussed methods in the literature survey, based on some relevant parameters such as Visual Scrambling, Correlation, Pixel Sensitivity, Key Space, Key sensitivity and NPCR.
Future works of this project would be towards improving the performance of the algorithm, understanding the patterns of pixels forming in the results and to assess whether there is a possible scope of threat to the security of this algorithm.

References

1) Ahmed, F., Siyal, M. Y., & Uddin Abbas, V. (2010). A secure and robust hash-based scheme for image authentication. Signal Processing, 90(5), 1456–1470
2) Image Authentication using Image Hashing based on Ring Partition with Corner Inclusion Ram Kumar Karsh, Himanshu Mishra, Tejveer Vashist and R. H. Laskar, 2018
3) Nag, A., Singh, J. P., Khan, S., Ghosh, S., Biswas, S., Sarkar, D., & Sarkar, P. P. (2011). Image encryption using affine transform and XOR operation. 2011 International Conference on Signal Processing, Communication, Computing and Networking Technologies. doi:10.1109/icsccn.2011.6024565
4) Tang, Z., Yang, Y., Xu, S., Yu, C., & Zhang, X. (2019). Image Encryption with Double Spiral Scans and Chaotic Maps. Security and Communication Networks, 2019, 1–15. doi:10.1155/2019/8694678
5) Amalarethinam, D. I. G., & Geetha, J. S. (2015). Image encryption and decryption in public key cryptography based on MR. 2015 International Conference on Computing and Communications Technologies (ICCCT). doi:10.1109/iccct2.2015.7292733
6) Image encryption using affine transform and XOR operation, Amitava nag, Jyoti Singh, Srabani khan, Saswati Ghosh Dept. of information technology, Academy of Technology Bandel, India.
7) Nagaraj, S., Raju, G. S. V. P., & Rao, K. K. (2015). Image Encryption Using Elliptic Curve Cryptography and Matrix. Procedia Computer Science, 48, 276-281.
8) Soleymani, A., Nordin, M. J., & Ali, Z. M. (2013). A Novel Public Key Image Encryption Based on Elliptic Curves over Prime Group Field. Journal of Image and Graphics, 1(1).
9) Deb, S., Biswas, B. & Bhuyan, B. Secure image encryption scheme using high efficiency word-oriented feedback shift register over finite field. Multimed Tools Appl 78, 34901–34925 (2019). https://doi.org/10.1007/s11042-019-08086-y

10) https://github.com/bjlittle/imagehash
11) https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation
12) https://pypi.org/project/Pillow/

13) https://matplotlib.org/users/pyplot_tutorial.html