SLICED IMAGE HANDLING AND ENCRYPTION

A REVIEW REPORT

Submitted by

ARKAJIT DATTA - 19BCI0115
ADRIJA MUKHOPADHYAY - 19BDS0159
MAMIDIBATHULA RISHIKA - 19BDS0163

Course Code: CSE4019
Course Title: IMAGE PROCESSING

Under the guidance of
Dr. Anisha M Lal
Professor
SCOPE, VIT, Vellore.

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

December-2021

TABLE OF CONTENTS

ABSTRACT

Image Security has become an important issue in data communication. Cryptography has come up as a solution, and plays an important role in information security systems. The purpose of the project is to demonstrate and develop a better and more secure system for image transfer across devices. This project is based on the research paper "Security for an Image using Bit-slice Rotation Method–image Encryption by R. Vijayaraghavan, S. Sathya and N. R. Raajan" [1].

Keywords: Bit plane slicing, Image encryption, Pseudo Random Generator

1. INTRODUCTION

1.1. Theoretical Background

Our project is developed in python and uses some comprehensive and well trusted algorithms like AES, ECC, and Bit Plane Slicing. We have also used a comprehensive way of transposition in the Bit Plane Slicing so as to make a more secure encryption algorithm.

1.2. Motivation

After reading some of the research papers and documentations of various ways for image handling on the web we concluded that the pre-existing systems are easy to attack if we use them individually. Hence, after some discussion with our professor Dr. Anisha M Lal we decided to further look into the field of Secure Image Transfer and try to develop a better and more secure algorithm.

2. LITERATURE SURVEY

	Title of the Paper And year	Algorithms used	Performance measures	Scope for future work
1	**Grid-based Image Encryption using RSA** April 2015	Grid-based encryption and decryption approach with RSA cryptosystem This approach takes the original image, performs block transformation followed by grid transformation, then finally a public key cryptosystem is applied	The original image is completely recovered from the encrypted image.	This paper's future work can include applying other asymmetric cryptosystems and in the case of division of images any symmetric cryptosystem. Additionally, it can be applied to audio and

		pixel-by-pixel to secure the image.		video encryption and decryption.
2	**Image Cryptography Using RSA Algorithm in Network Security September 2015**	RSA algorithm is used to encrypt the image files to enhance the security in the communication area for data transmission.	In this paper the cryptography mechanism using the RSA algorithm with the public key encryption is to increase the security levels of the encrypted data. Here one key is needed to encrypt and another key is needed to decrypt the image. Finally, the image cryptography experiment provides the feasibility of security to the image in network security. The data is not viewed by no one without the knowledge of cryptography.	The research done by the authors in the paper can be used further, to improve encryption techniques to increase security.
3	**Digital Image Encryption Based on RSA Algorithm (Jan. 2014)**	comparison between the RSA cryptosystem which is one of asymmetric cryptosystems and the two symmetric systems, DES and Blowfish.	The results showed that time taken using mathematical relations in RSA make steps faster implemented than DES and Blowfish algorithms and with more secured data than symmetric systems	In RSA, the value of chosen prime numbers Q and P controls time in key generation so that it increases time taken due to making it more secure than before. So , future work is to reduce time with more security
4	**Image Encryption using Combination of Chaotic System and Rivers Shamir Adleman (RSA) August 2015**	combining the two algorithms namely Chaos-based algorithms and RSA algorithm into one application.	Experiments conducted show that the proposed algorithm possesses robust security features such as fairly uniform distribution, high sensitivity to both keys and plain images, almost ideal entropy, and the ability to highly de-correlate adjacent pixels in the cipher images. Furthermore, it has a large key space, and transforms images to pure text files which greatly increases its security for image encryption applications. The proposed algorithm takes a long time to complete an encryption process compared to other methods. This is because the encryption process will transform a plain image into a base text cipher. The other methods do not perform transformations on the generated cipher.	For future work, they will try to develop algorithms that can perform encryption on all image sizes and all types of images.
5	**An Ethical Way of Image Encryption Using ECC (2009)**	Elliptic Curve Cryptography	It has been observed that RSA is far too slow compared to ECC. The researchers have encrypted each and every pixel using ECC. Elliptic curve ciphers use very small key sizes and computationally is very e4cient. N. Koblitz and Miller independently proposed the elliptic curve cryptosystem. The Performance achieved in ECC was great with the computer taking 0.03 secs to 0.35 secs to encrypt the images, with image ranging from 256*256 to 512*512.	Some other texture models such as Gaussian Mrf models, fractal models, Gabor filters, and time series models may be used instead of Using Mrf for information concealment under our proposed framework.
6	**Simulation of Image Encryption using AES Algorithm**	Advanced Encryption Standard	Each round has 4 operations and it is iterative in nature.So the output of first round is fed to the second round as input data and perform the same operations with another set of keys.This process continued until the last round reach.In the last round,there is no mixcolumn operation.The State array	The Authors referred to using asymmetric algorithms for further encryption processes.

			obtained after the last round is the required cipher text for transmission	

3. OVERVIEW

3.1. Aim of the proposed Work

The project is aimed to be more secure than the pre-existing image transfer algorithm that exist in the current industry. We will try to encrypt the image using a novel algorithm which includes slicing of the image (using bit-plane slicing).

3.2. Objective(s) of the proposed work

Following are the Objectives of the project:

I. To be more secure than other algorithms.

II. To be able to receive the same image without any significant loss in the image quality.

III. Help us understand the concepts practically and in depth.

3.3. Introduction and Related Concepts

Our Project Uses a few algorithms like AES, ECC, and Bit Plane Slicing.

AES (ADVANCED ENCRYPTION STANDARD):

The Advanced Encryption Standard (AES), also known by its original name Rijndael, is a specification for the encryption of electronic data established by the U.S. National Institute of Standards and Technology (NIST) in 2001.

ECC:

Elliptic Curve Cryptography (ECC) is a key-based technique for encrypting data. ECC focuses on pairs of public and private keys for decryption and encryption of web traffic.

BIT PLANE SLICING:

Bit plane slicing is a method of representing an image with one or more bits of the byte used for each pixel. One can use only MSB to represent the pixel, which reduces the original gray level to a binary image. The three main goals of bit plane slicing is:

- Converting a gray level image to a binary image.

- Representing an image with fewer bits and corresponding the image to a smaller size
- Enhancing the image by focusing

TRANSPOSITION OF SLICED IMAGES:

Transposition of the Sliced Images is an algorithm that we have used to shuffle the image parts obtained after slicing. This ensures more safety as it's an extra layer like a jigsaw puzzle.

3.4.Framework, Architecture

ENCRYPTION

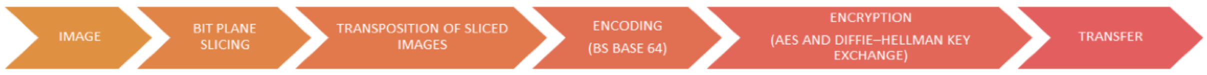
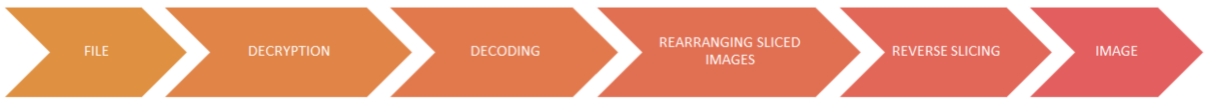


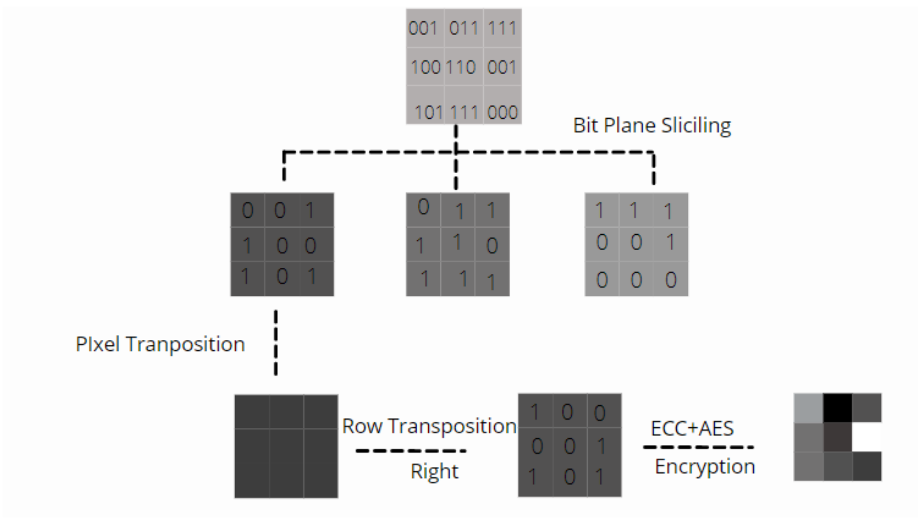
IMAGE
BIT PLANE SLICING
TRANSPOSITION OF SLICED IMAGES
ENCODING (BS BASE 64)
ENCRYPTION (AES AND DIFFIE–HELLMAN KEY EXCHANGE)
TRANSFER

DECRYPTION

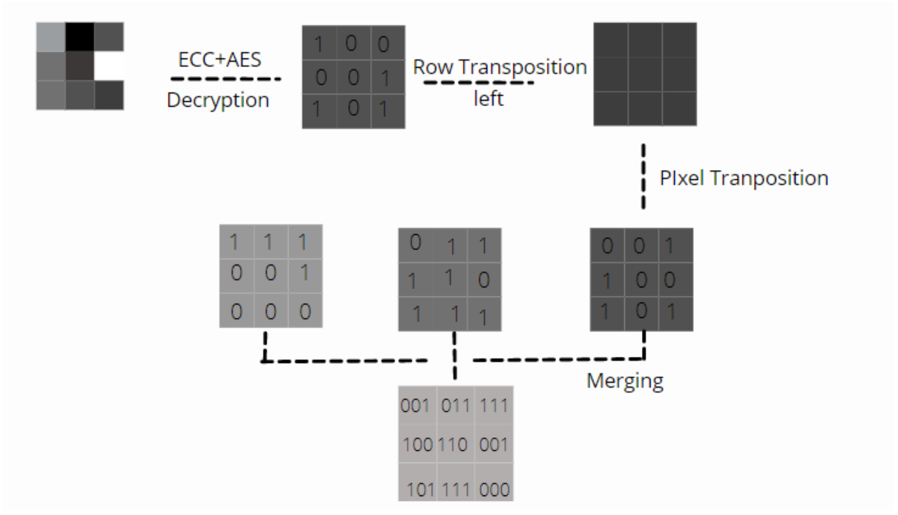


LOW LEVEL DESIGN:

ENCRYPTION



DECRYPTION



THEORETICAL FRAMEWORK:

Seed Function:

We are first generating a sequence of large random numbers, but for the convenience of the example, let’s say we generated a sequence 13, 14, 9 and so on.

Bit Plane slicing:

We are performing 8 bits slicing on our image. We know images are made of pixels, each pixel has bytes and those bytes have corresponding bits. When we slice an image, we present the image in the form of those corresponding bits. For the convenience of the example, we are performing 3 bits slicing, where we are splitting a 3x3 matrix into 3 3x3 matrices, like if we a 001, we are splitting it as 0, 0 and 1. For decryption, we merge the matrices in a way that they result into the original matrix.

Pixel Transposition:

- **Row transposition:**

  Now we are performing row transposition. What we are doing here is, assigning different random seed values that we generated, to each row. When a random seed value is assigned to a row, we will perform a mod operation to keep it in the range. And then right shifting the row by that random number. For example, we have a 3 by 3 matrix, and random values 13, 14, 9 are assigned to each row respectively. So now the column number is 3 only, so for each row, we will do 13 mod 3, 14 mod 3, and 9 mod 3, now the values assigned to each row are 1, 2 and 0. So, the first row, the pixels will be right shifted by 1 steps, 2nd will be shifted by 2 steps and 3rd will be by 0 steps. Now the 3 by 3 matrix was just an example. In reality, we might be having a 512 x 512 matrix, or 768 x 768. So, we will be performing row transposition with random seed values in this manner to make the intermediate image during encryption more gibberish in order to make the system more secure.

For decryption, we perform circular left shift with the same mod values.

ECC AES Hybrid:

After bit plane slicing each sliced image is being encrypted. there is a unique key for each image which is being further encrypted using ECC (hybrid with AES)

3.5. Proposed System Model

METHODOLOGY

1. Input the image.
2. Seed Function is used to generate a sequence of random numbers according to the size of the image.
3. Bit Plane Slicing is performed on the input image
4. For each Bit plane sliced image perform the following function: -

    - Transposition of bits Pixel wise (circular right shift)
    - Row wise right shift.

5. Send the original message and the encrypted msg to the user.
6. Private keys are decrypted, which are used to decrypt each of the bit plane sliced images.
7. Decryption goes as follows: -

    Using the same seed function, we will generate the same key sequence and get the original image back

    The following functions are performed

    - Row wise circular left shift
    - Transposition of pixels (bitwise) - circular left shift.

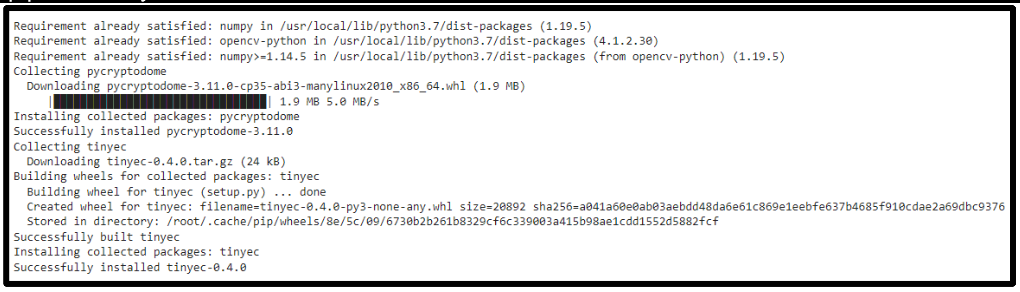
8. Merging all the 8-bit sliced images is performed, to get the final output image.

4. SYSTEM ANALYSIS AND DESIGN

IMPLEMENTATION:

This Section fetches all dependencies need for this project to function correctly

```
!pip install numpy
!pip install opencv-python
!pip install pycryptodome
!pip install tinyec
```



```
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (1.19.5)
Requirement already satisfied: opencv-python in /usr/local/lib/python3.7/dist-packages (4.1.2.30)
Requirement already satisfied: numpy>=1.14.5 in /usr/local/lib/python3.7/dist-packages (from opencv-python) (1.19.5)
Collecting pycryptodome
  Downloading pycryptodome-3.11.0-cp35-abi3-manylinux2010_x86_64.whl (1.9 MB)
     |████████████████████████████████| 1.9 MB 5.0 MB/s
Installing collected packages: pycryptodome
Successfully installed pycryptodome-3.11.0
Collecting tinyec
  Downloading tinyec-0.4.0.tar.gz (24 kB)
Building wheels for collected packages: tinyec
  Building wheel for tinyec (setup.py) ... done
  Created wheel for tinyec: filename=tinyec-0.4.0-py3-none-any.whl size=20892 sha256=a041a60e0ab03aebdd48da6e61c869e1eebfe637b4685f910cdae2a69dbc9376
  Stored in directory: /root/.cache/pip/wheels/8e/5c/09/6730b2b261b8329cf6c339003a415b98ae1cdd1552d5882fcf
Successfully built tinyec
Installing collected packages: tinyec
Successfully installed tinyec-0.4.0
```

Bit Plane Slicing Testing

Importing all Necessary Libraries

```
#imports
import numpy as np
import cv2
import matplotlib.pyplot as plt
from google.colab.patches import cv2_imshow
from PIL import Image
import random
import sys
```

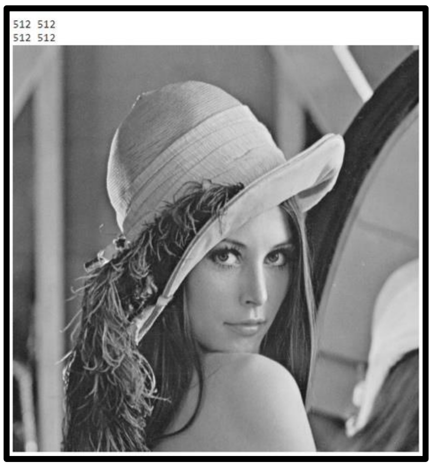
SeedRandom() generates a random sequence of numbers

```
class SeedRandom():
   def __init__(self,seed):
      random.seed(seed)
   def rand(self):
      return int(random.random()*(10 ** 16)) #function to generate key
# seed_obj = SeedRandom(90)
# for i in range(100):
#   print(seed_obj.rand(8))
```

Importing the input Image and Displaying the same

```
img = cv2.imread(r'/content/lena.jpg',0) #input image
print(img.shape[0],img.shape[1])
final_x = img.shape[0]
final_y = img.shape[1]
print(final_x,final_y)
#display image
```

```
cv2_imshow(img)
```



Functions that will be used for bit plane slicing

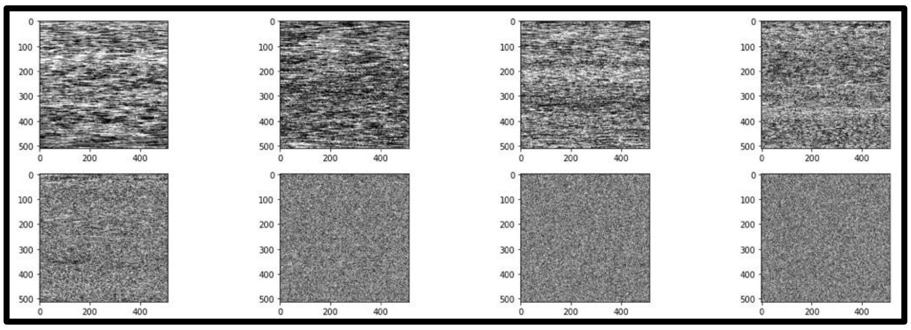
1. Transposition of bits Pixel wise (circular right shift)
2. Row wise right shift

```
list_seeds = []
def cov_binary(num):
   binary_num = [int(i) for i in list('{0:0b}'.format(num))]
   for j in range(8 - len(binary_num)):
      binary_num.insert(0,0)
   return binary_num
def conv_decimal(listt):
   x = 0
   for i in range(8):
      x = x + int(listt[i])*(2**(7-i))
   return x
def discriminate_bit(bit,img):
   seed = int(random.random()*9223372036854775807) #random key
   list_seeds.append(seed)
   seed_obj = SeedRandom(seed) #seed number passed (key is passed to the seed function)
   height_row = img.shape[0]
   width_col = img.shape[1]
   list_seq = [] #list stores the sequence
   for i in range(height_row*width_col):
    list_seq.append(seed_obj.rand()) #constantly generate the random number in sequence -- 250000
   z = np.zeros([height_row,width_col])
   final_image = np.zeros([height_row,width_col])
   temp = 0
   for i in range(height_row):
      for j in range(width_col):
         x = cov_binary(img[i][j])
         for k in range(8):
            if k == bit:
               #performing right shit
               shift = (list_seq[temp]%8) + k
               if shift > 7:
```

```
         shift = shift - 8
        #value = x[k] #storing the value for shift
        x[k] = x[k]
        temp = temp+1
      else:
        x[k] = 0
    #x[shift] = value
    x1 = conv_decimal(x)
    z[i][j] = x1
temp = 0
for i in range(height_row):
 for j in range(width_col):
  new_col = j+(list_seq[temp]%final_y)
  if new_col >= width_col:
   new_col = new_col - (width_col)
  #print(new_col)
  final_image[i][new_col] = z[i][j]
 temp = temp + 1
return final_image
```

Display all bit plane sliced images

```
fig = plt.figure()
fig.set_figheight(20)
fig.set_figwidth(20)
encrypted_images_list = []
for i in range(1,9):
  fig.add_subplot(6,4,i)
  display_image = discriminate_bit(i-1,img)
  encrypted_images_list.append(display_image) #saving the encrypted images
  path = r"/content/"+str(i)+"_encrypted.jpg"
  cv2.imwrite(path,display_image)
  plt.imshow(display_image, cmap='gray')

#plt.show(block=True)
```



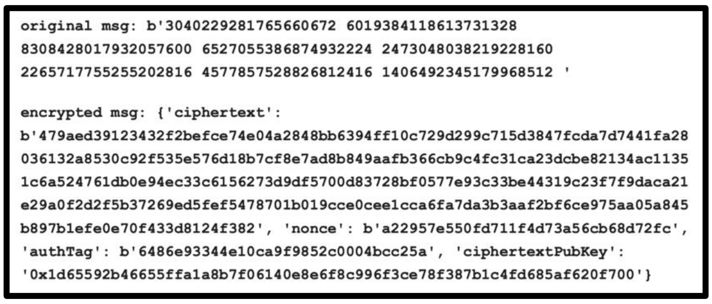
ECC + AES hybrid

```
def encrypt_AES_GCM(msg, secretKey):
    aesCipher = AES.new(secretKey, AES.MODE_GCM)
    ciphertext, authTag = aesCipher.encrypt_and_digest(msg)
    return (ciphertext, aesCipher.nonce, authTag)

def decrypt_AES_GCM(ciphertext, nonce, authTag, secretKey):
    aesCipher = AES.new(secretKey, AES.MODE_GCM, nonce)
    plaintext = aesCipher.decrypt_and_verify(ciphertext, authTag)
    return plaintext

def ecc_point_to_256_bit_key(point):
    sha = hashlib.sha256(int.to_bytes(point.x, 32, 'big'))
    sha.update(int.to_bytes(point.y, 32, 'big'))
    return sha.digest()

def encrypt_ECC(msg, pubKey):
    ciphertextPrivKey = secrets.randbelow(curve.field.n)
    sharedECCKey = ciphertextPrivKey * pubKey
    secretKey = ecc_point_to_256_bit_key(sharedECCKey)
    ciphertext, nonce, authTag = encrypt_AES_GCM(msg, secretKey)
    ciphertextPubKey = ciphertextPrivKey * curve.g
    return (ciphertext, nonce, authTag, ciphertextPubKey)

def decrypt_ECC(encryptedMsg, privKey):
    (ciphertext, nonce, authTag, ciphertextPubKey) = encryptedMsg
    sharedECCKey = privKey * ciphertextPubKey
    secretKey = ecc_point_to_256_bit_key(sharedECCKey)
    plaintext = decrypt_AES_GCM(ciphertext, nonce, authTag, secretKey)
    return plaintext
```

Tinyec library to generate a ECC private-public key

```
from tinyec import registry
from Crypto.Cipher import AES
import hashlib, secrets, binascii

curve = registry.get_curve('brainpoolP256r1')#curve we have chosen
message = ""
for i in list_seeds:
  message = message+str(i)
  message = message+" "
msg = b'Text to be encrypted by ECC public key and '\
      b'decrypted by its corresponding ECC private key' #messsage which needs to be worked upon
res = bytes(message,'utf-8') #converted it to bytes
msg = res
print("original msg:", msg)
privKey = secrets.randbelow(curve.field.n)
pubKey = privKey * curve.g
```

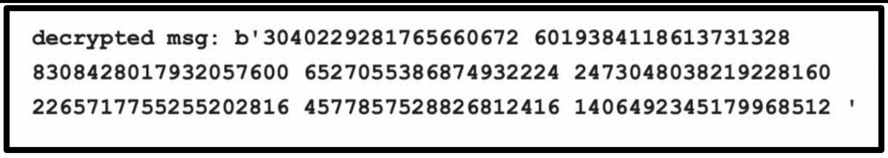
```
encryptedMsg = encrypt_ECC(msg, pubKey)
encryptedMsgObj = {
    'ciphertext': binascii.hexlify(encryptedMsg[0]),
    'nonce': binascii.hexlify(encryptedMsg[1]),
    'authTag': binascii.hexlify(encryptedMsg[2]),
    'ciphertextPubKey': hex(encryptedMsg[3].x) + hex(encryptedMsg[3].y % 2)[2:]
}
print("encrypted msg:", encryptedMsgObj)
```



```
original msg: b'3040229281765660672 6019384118613731328
8308428017932057600 6527055386874932224 2473048038219228160
2265717755255202816 4577857528826812416 1406492345179968512 '

encrypted msg: {'ciphertext':
b'479aed39123432f2befce74e04a2848bb6394ff10c729d299c715d3847fcda7d7441fa28
036132a8530c92f535e576d18b7cf8e7ad8b849aafb366cb9c4fc31ca23dcbe82134ac1135
1c6a524761db0e94ec33c6156273d9df5700d83728bf0577e93c33be44319c23f7f9daca21
e29a0f2d2f5b37269ed5fef5478701b019cce0cee1cca6fa7da3b3aaf2bf6ce975aa05a845
b897b1efe0e70f433d8124f382', 'nonce': b'a22957e550fd711f4d73a56cb68d72fc',
'authTag': b'6486e93344e10ca9f9852c0004bcc25a', 'ciphertextPubKey':
'0x1d65592b46655ffa1a8b7f06140e8e6f8c996f3ce78f387b1c4fd685af620f700'}
```

Decryption of private key

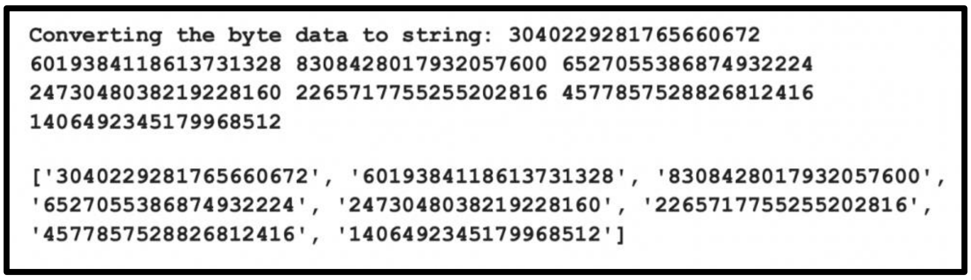
```
decryptedMsg = decrypt_ECC(encryptedMsg, privKey)
print("decrypted msg:", decryptedMsg)
```



```
decrypted msg: b'3040229281765660672 6019384118613731328
8308428017932057600 6527055386874932224 2473048038219228160
2265717755255202816 4577857528826812416 1406492345179968512 '
```

Decryption of the Image

```
#convert the byte data to string
decrypted_str = decryptedMsg.decode('UTF-8')
print("Converting the byte data to string: "+decrypted_str)
#splitting the string to get the keys
list_decrypted_keys = decrypted_str.split(" ")
list_decrypted_keys.pop()
print(list_decrypted_keys) #this list will contain all the keys
```

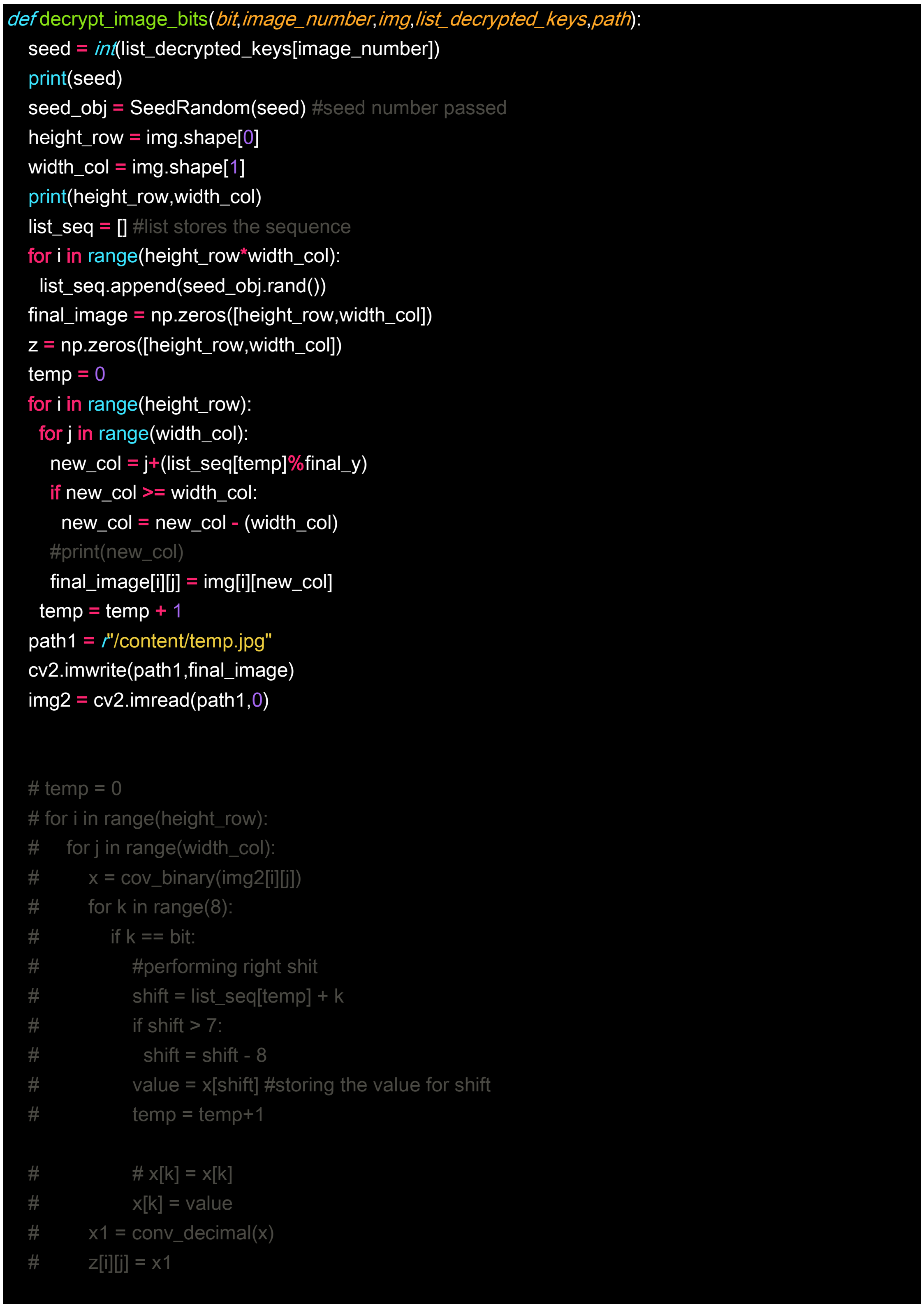


```
Converting the byte data to string: 3040229281765660672
6019384118613731328 8308428017932057600 6527055386874932224
2473048038219228160 2265717755255202816 4577857528826812416
1406492345179968512

['3040229281765660672', '6019384118613731328', '8308428017932057600',
'6527055386874932224', '2473048038219228160', '2265717755255202816',
'4577857528826812416', '1406492345179968512']
```

Same seed function we will generate the same key sequence and get the original image back.

Following operations are performed to decrypt the images

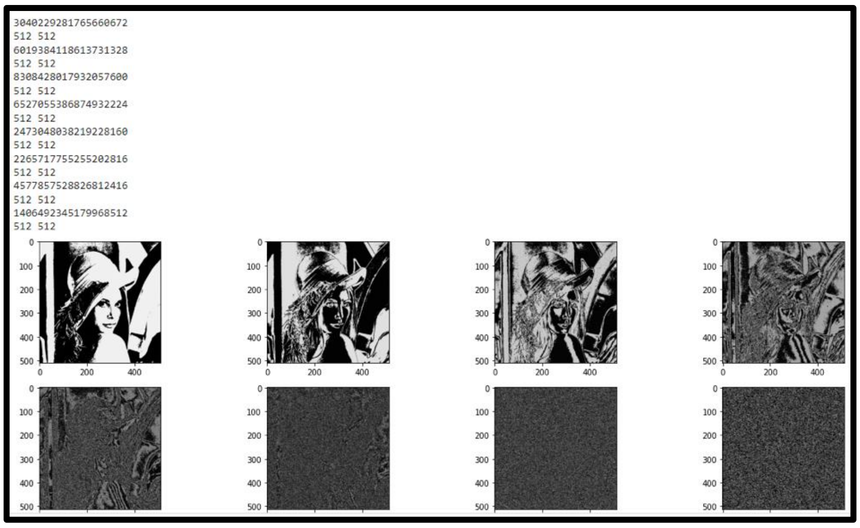
1. Row wise circular left shift
2. Transposition of pixels (bitwise) - circular left shift



```
def decrypt_image_bits(bit,image_number,img,list_decrypted_keys,path):
  seed = int(list_decrypted_keys[image_number])
  print(seed)
  seed_obj = SeedRandom(seed) #seed number passed
  height_row = img.shape[0]
  width_col = img.shape[1]
  print(height_row,width_col)
  list_seq = [] #list stores the sequence
  for i in range(height_row*width_col):
    list_seq.append(seed_obj.rand())
  final_image = np.zeros([height_row,width_col])
  z = np.zeros([height_row,width_col])
  temp = 0
  for i in range(height_row):
    for j in range(width_col):
      new_col = j+(list_seq[temp]%final_y)
      if new_col >= width_col:
        new_col = new_col - (width_col)
      #print(new_col)
      final_image[i][j] = img[i][new_col]
    temp = temp + 1
  path1 = r"/content/temp.jpg"
  cv2.imwrite(path1,final_image)
  img2 = cv2.imread(path1,0)


  # temp = 0
  # for i in range(height_row):
  #     for j in range(width_col):
  #         x = cov_binary(img2[i][j])
  #         for k in range(8):
  #             if k == bit:
  #                 #performing right shit
  #                 shift = list_seq[temp] + k
  #                 if shift > 7:
  #                   shift = shift - 8
  #                 value = x[shift] #storing the value for shift
  #                 temp = temp+1

  #                 # x[k] = x[k]
  #                 x[k] = value
  #         x1 = conv_decimal(x)
  #         z[i][j] = x1
  return final_image
  # for i in range(height_row):
  #   for j in range(width_col):
  #     new_col = j - (list_seq[temp]*50)
  #     print(new_col)
  #     if new_col < 0:
  #       new_col = width_col + new_col
  #     print(new_col)
  #     #print(new_col)
  #     final_image[i][new_col] = z[i][j]
  #   temp = temp + 1
  # return final_image
```

Displaying decrypted 8 bit plane sliced images

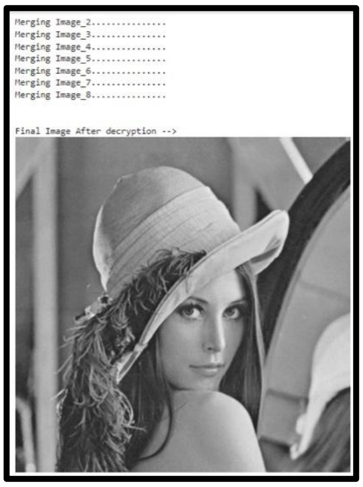
```
fig = plt.figure()
fig.set_figheight(20)
fig.set_figwidth(20)
encrypted_images_list = []
for i in range(1,9):
  fig.add_subplot(6,4,i)
  path = r"/content/"+str(i)+"_encrypted.jpg"
  img = cv2.imread(path,0)
  display_image = decrypt_image_bits(i-1,i-1,img,list_decrypted_keys,path)
  plt.imshow(display_image, cmap='gray')

plt.show(block=True)
```



```
3040229281765660672
512 512
6019384118613731328
512 512
8308428017932057600
512 512
6527055386874932224
512 512
2473048038219228160
512 512
2265717755255202816
512 512
4577857528826812416
512 512
1406492345179968512
512 512
```

Merging all the images to get the Final Output

```
def final_output_generation(height_row,width_col):
  #final_output_image = np.zeros([height_row,width_col])
  #path_temp = r"/content/temp1.jpg"
  #cv2.imwrite(path_temp,final_output_image)
  final_output_image = cv2.imread('/content/1_decrypted.jpg',0)
  for b in range(2,9):
    path = "/content/"+str(b)+"_decrypted.jpg"
    img = cv2.imread(path,0)
    print("Merging Image_"+str(b)+"...............")
    for i in range(height_row):
        for j in range(width_col):
            final_output_image[i][j] += img[i][j]
  return final_output_image

display = final_output_generation(final_x,final_y)
cv2.imwrite("Final_image_output.jpg",display)
img = cv2.imread(r"/content/Final_image_output.jpg",0)
print("\n")

print("Final Image After decryption --> ")
cv2_imshow(img)
#plt.imshow(display, cmap='gray')
```



```
Merging Image_2...............
Merging Image_3...............
Merging Image_4...............
Merging Image_5...............
Merging Image_6...............
Merging Image_7...............
Merging Image_8...............


Final Image After decryption -->
```

5. COMPARISON AND PERFORMANCE EVALUATION

The bit plane slicing is quite used for the image encryption also there are quite many algorithms which use bit plane slicing for image encryption. Here in the comparison, we are comparing our proposed method with another algorithm which takes in account a scrambling algorithm and rotation of the pixels in order to encrypt the image. In the proposed method by the researchers, they have used a dynamic rotation function which rotates each bit planes with a certain angle.

1st bit plane [Rotated by 90°] – $\begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$

2nd bit plane [Rotated by 180°] – $\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$

3rd bit plane [Rotated by 270°] – $\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$

4th bit plane [Rotated by 90°] – $\begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}$

5th bit plane [Rotated by 180°] – $\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$

6th bit plane [rotated by 270°] – $\begin{bmatrix} 0 & 0 \\ 1 & 1 \end{bmatrix}$

The above figure shows how the images are being rotated.

Limitations found:

- Here the angles of rotation can be only 90,180 and 270. As images are matrices they can’t be rotated with any other angle. Hence this makes the system really predictable to the brute force attacks.

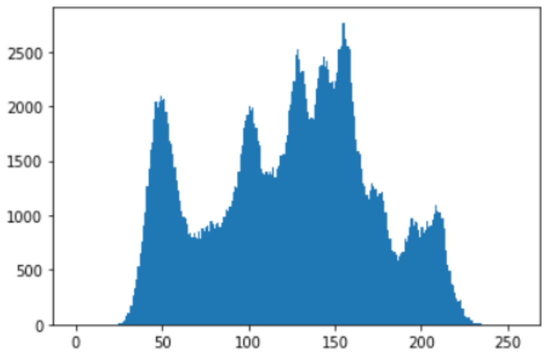
Our encryption algorithm is more dynamic and much more secured compared to given scrambling algorithm. We have 2 Tier encryption and decryption standards with a 16-digit key for each image.

Pros of our proposed methodology –

- After bit plane slicing, the algorithm does a pixel-by-pixel transposition for which we generate a unique random number for each pixel. Let’s us say we are encrypting a 512 X 512 image. For this image our key generator generated more that 2,50,000 unique keys which nearly omits out the scope of any brute force attack which was earlier possible.

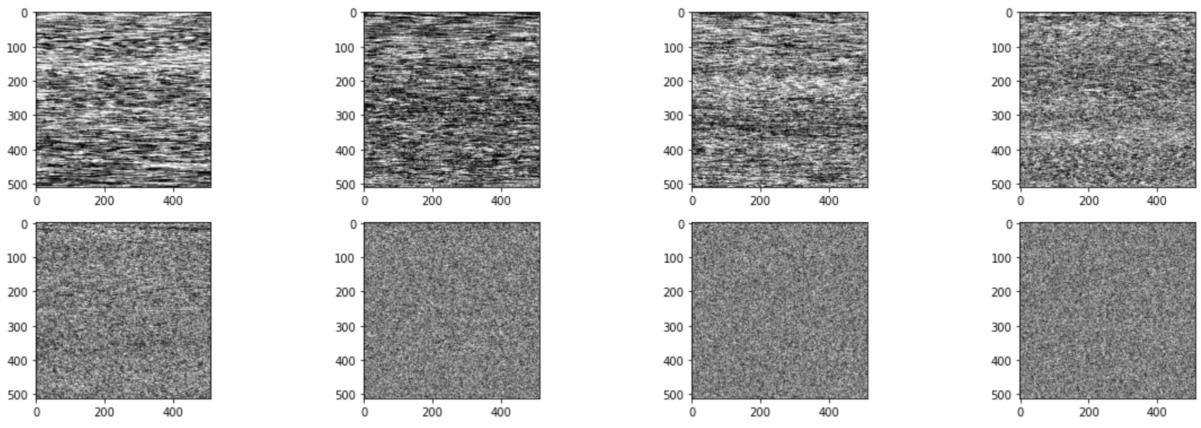
- Next the algorithm performs a row wise scrambling for which we have unique key for every row. This in-turn increases the complexity and security of the image encryption as the hacker needs to try ‘x’ number of keys for every row. This indeed multiplies the complexity with a huge value of x! .
- We also encrypt our keys with ECC+AES which makes our key transfer much more secured and nearly impossible to hack in.

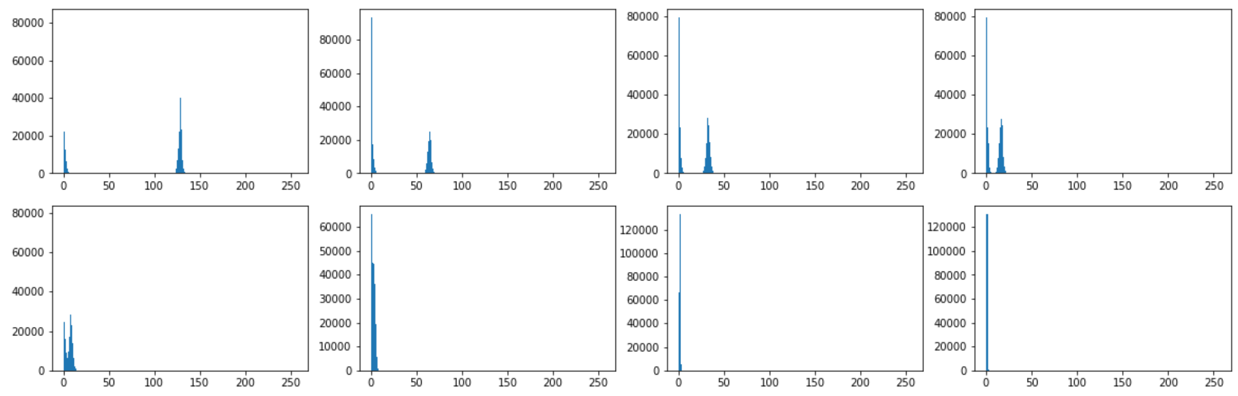
Further we have generated the histogram and the PSNR number of our original and encrypted image in order to compare with the one of the existing methods.



(a) (b)

Here Figure a is the input image and the figure b is the plotted histogram of the same image. Further we will be showing the histogram of the encrypted images.

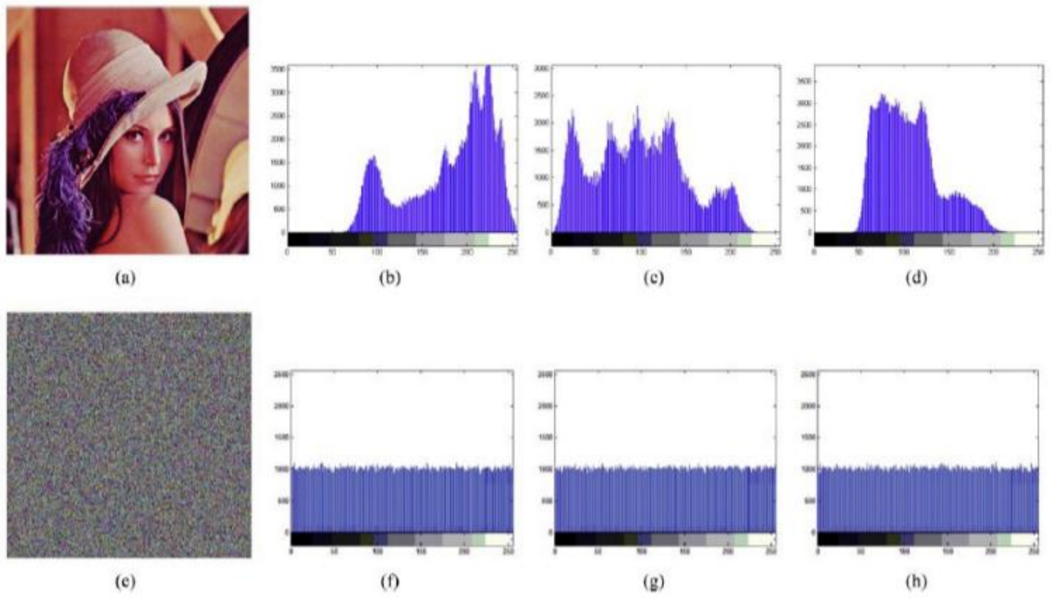




(c)

Here the plot c shows the respective histograms of the encrypted images. Here we can see the achieved histograms have no relation with the one given as in the input, moreover the histograms show that the pixels are located to certain same level of gray level shading. This makes the system more secure and shows the validation of its complexity.

Histogram analysis of the compared image encryption algorithm -



6. RESULTS AND DISCUSSION

Encryption and Decryption is achieved to a good extent. The proposed algorithm takes approximately 3 seconds for encrypting one image. We have achieved a good level of decryption without much loss. In the worst cases this algorithm takes O(n2). In our case, with a 512X512 image we were able to encrypt in 20 seconds, and decrypt the same in 5 seconds.

For now, this project focuses on grayscale images. Improving this project for colour images is a future scope for this project.

7. REFERENCES

WEBLINKS:

1. https://sciresol.s3.us-east-2.amazonaws.com/IJST/Articles/2014/Issue-Supplementary-4/Article1.pdf

JOURNAL:

1. Singh, B. K., & Gupta, S. K. (2015). Grid-based image encryption using RSA. International Journal of Computer Applications, 115(1).
2. Anandakumar, S. (2015). Image cryptography using RSA algorithm in network security. International Journal of Computer Science & Engineering Technology, 5(9), 326-330.
3. El-Deen, A., El-Badawy, E., & Gobran, S. (2014). Digital image encryption based on RSA algorithm. J. Electron. Commun. Eng, 9(1), 69-73.
4. Irfan, P., Prayudi, Y., & Riadi, I. (2015). Image encryption using combination of chaotic system and rivers shamir adleman (RSA). International Journal of Computer Applications, 123(6).
5. Kalubandi, V. K. P., Vaddi, H., Ramineni, V., & Loganathan, A. (2016, October). A novel image encryption algorithm using AES and visual cryptography. In 2016 2Nd international conference on next generation computing technologies (NGCT) (pp. 808-813).IEEE.
6. Gupta, K., Silakari, S., Gupta, R., & Khan, S. A. (2009, July). An ethical way of image encryption using ECC. In 2009 First International Conference on Computational Intelligence, Communication Systems and Networks (pp. 342-345). IEEE.
7. Karthigaikumar, P., & Rasheed, S. (2011). Simulation of image encryption using AES algorithm. IJCA special issue on “computational science-new dimensions & perspectives” NCCSE, 166-172.
8. Radhadevi, P., & Kalpana, P. (2012). Secure image encryption using AES. International Journal of Research in Engineering and Technology, 1(2), 115-117.
9. Zeghid, M., Machhout, M., Khriji, L., Baganne, A., & Tourki, R. (2007). A modified AES based algorithm for image encryption. International Journal of Computer Science and Engineering, 1(1), 70-75.