SECURE IMAGE TRANSMISSION SYSTEM USING MODIFIED AES

Project – Report

CSE3501 – Information Security Analysis and Audit

Submitted By:

Nitin Kumar (20BIT0031)

Mukul Chavan (20BIT0238)

Rohan B (20BIT0398)

Rohil Saxena(20BIT0404)

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

Contents

Abstract

The older picture encryption techniques are no longer effective with modern technologies. Image encryption is becoming a target for several attacks. Data transmission and storage are becoming more and more dependent on information security. Many procedures often employ images.Therefore, it is crucial to safeguard picture data against unwanted access.Therefore, in order to send images utilising a variety of algorithms and by tweaking AES, we devised an image transmission system. This gave us a far more effective and secure end result.

Introduction

The handling of picture transfers using current technologies is no longer safe and capable of withstanding significant assaults. In order to provide a safer and more effective platform for picture transfer, we are putting out a new system with updated AES. The three main security objectives are, as we all know, confidentiality, integrity, and availability. We made an effort to address all of these objectives in our secure transmission method.

We carefully analysed a number of picture encryption techniques before deciding to focus on AES. We then conducted more research on AES and looked for its flaws. We came to understand that the mix column operation and time complexity were the two constraints. Utilizing a constant matrix, mix column performs straightforward matrix multiplication. The speedier but more difficult mathematical process known as bit permutation has been used in place of the mix column operation.

Prior to AES, we are conducting 8-bit pixel slicing and shuffling to further increase security.

Problem Statement

The main problems that arise during image transmission process are with respect to the time it takes to reach the destination and its security level. For real time image encryption only those ciphers are preferable which takes lesser amount of computational time. When an original image is been transmitted from one end to another over a network, security is essential. In order to secure the data which we send it has to be encrypted. So that the intruder would not get to know or the data cannot be hacked. The images are split and combined, at the decryption end using the same techniques original images are obtained. Hence, the image transmission takes place safely.

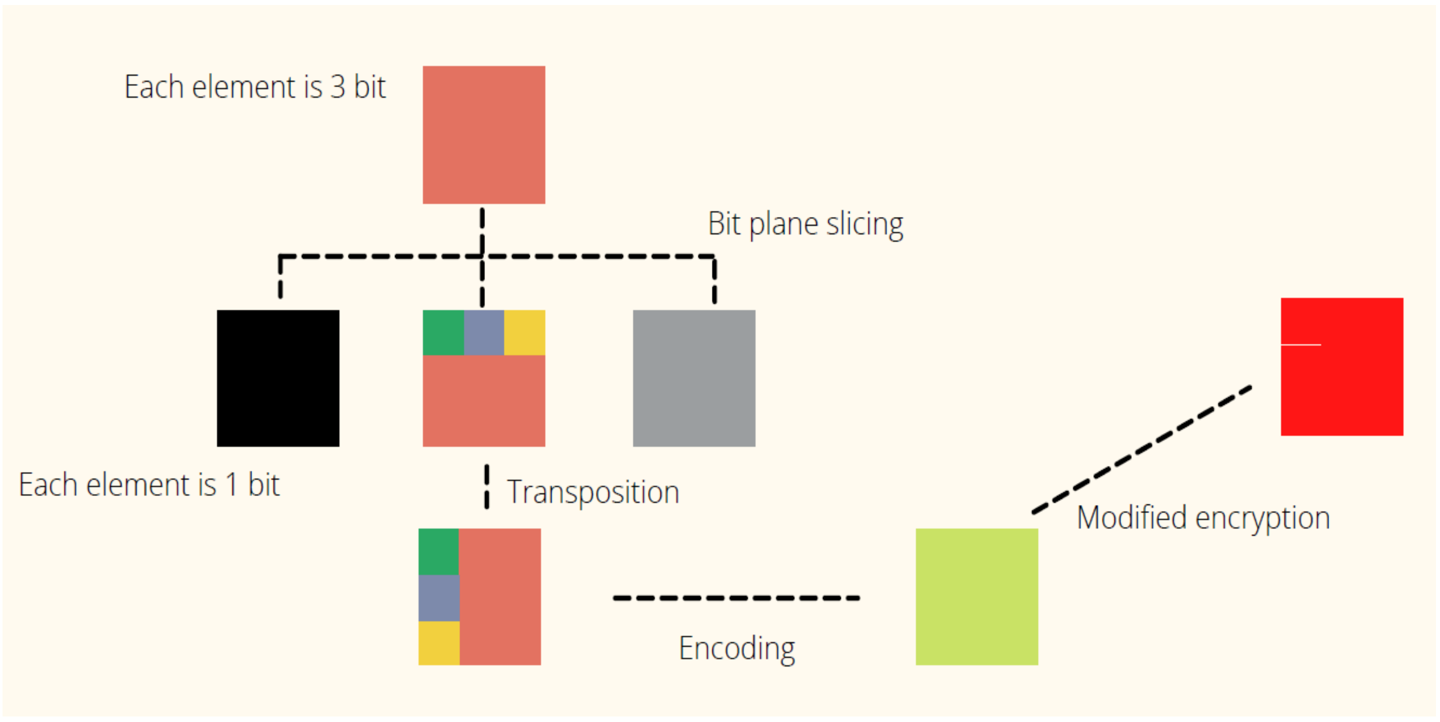
Literature Survey

	Title of the Paper And Author	Research findings	What did we address
1	Modified AES for Text and Image Encryption Heidilyn V. Gamido, Ariel M. Sison, Ruji P. Medina	The research modified the standard AES by using bit permutation technique instead of the Mix Columns Transformation. The use of a permutation technique in an encryption algorithm achieved a minimum encryption time, decreased memory requirement, higher avalanche effect and is easy to implement.	To increase security we hope to add more encryption techniques before applying the modified AES algorithm.
2	A New Image Encryption Algorithm Based Slicing and Displacement Followed By Symmetric and Asymmetric Cryptography techniques Ankita Maheshwari	A strong concept has been proposed for image security using a new technique based on the slicing of the image into sub parts and displacement of pixel position of each sub part.	The proposed research resolving and improving security issue of existing algorithm and improving overall efficiency by adding more confusion with the help of proposed encryption algorithm thus resulting in a strong cryptographic algorithm
3	SECURE IMAGE ENCRYPTION USING AES P. Radhadevi, P. Kalpana	This approach offers enhanced security, it aims to provide user satisfaction by transmitting personal and sensitive image data securely. The Advanced Encryption Standard offers the flexibility of allowing different key sizes 128 bit,192 bit and 256-bit key and the security is based on the various random key selections, different S-box and strong transformations..	This paper presents an application of AES (Advanced Encryption Standard) operations in image encryption and decryption. The encrypted cipher images always display the uniformly distributed RGB pixels.
4	A Modified AES Based Algorithm for Image Encryption M. Zeghid, M. Machhout, L. Khriji, A. Baganne, and R. Tourki	We add a key stream generator (A5/1, W7) to AES to ensure improving the encryption performance; mainly for images characterized by reduced entropy. The implementation of both techniques has been realized for experimental purposes. Detailed results in terms of security analysis and implementation are given. Comparative study with traditional encryption algorithms is shown the superiority of the modifiedalgorithm.	To increase protection we are hopingto add greater encryption techniquesbefore making use of the modified AES algorithm
5	Image Encryption And Decryption Using Blowfish Algorithm In Matlab 2013	This paper [14] proposed encryption and decryption of images using a secret-key block cipher called 64-bits blowfish designed to increase security and to improve performance. This algorithm is used as a variable key size up to 448 bits. It employs feistel network which iterates simple function 16 times. The blowfish algorithm is safe against unauthorized at-tack and runs faster than the popular existing algorithms.	The proposed algorithm is designed and realized using MATLAB. Hence if the number of rounds are increased then the blowfish algorithm becomes stronger. Since Blowfish does not have any known security weak points so far it can be considered as an excellent standard encryption algorithm.

6	**Developing a secured image file management system using modified AES** *Heidilyn V. Gamido, Marlon V. Gamido, Ariel M. Sison*	The shared image files are stored in the server in an encrypted manner to provide additional security to the owner of the file. A modified AES algorithm using bit permutation was used to encrypt the image files. Based on the experimental result, image files were successfully encrypted in the server and can only be decrypted by the intended recipient of the file providing an efficient and reliable way of exchanging images.	This paper introduce an image file management system that can be used to store, view , share images and provide additional protection to the owner of the image. The encrypted image is received, viewed and downloaded only by the intended recipient.
7	**Securing Medical Images by Image Encryption using Key Image** Shrija Somaraj , Mohammed Ali Hussain	The security of images has attracted more attention recently, and many different image encryption methods have been proposed for enhancing the security of images. They have presented image encryption technique using the Hill cipher method. A modification of the Advanced Encryption Standard which presents a high level of security and better image encryption.	There are three kinds of encryption techniques namely substitution, transposition or permutation and techniques that include both transposition and substitution. Substitution schemes change the pixel values while permutation schemes just shuffle the pixel values based on the algorithm. In some cases both the methods are combined to improve security. So we are proposing the high security during transmission.
8	A Survey On Different Image Encryption and Decryption Techniques Rinki Pakshwar, Vijay Kumar Trivedi, Vineet Richhariya	This paper focuses mainly on the different kinds of image encryption and decryption techniques. In addition focuses on image encryption techniques, As the use digital techniques for transmitting and storing images are increasing, it becomes an important issue that how to protect the confidentiality, integrity and authenticity of images.	This helped us get an idea about Image encryption and decryption techniques and how popular these are in general media
9	Analytical Study of Hybrid Techniques for Image Encryption and Decryption Chiranj Lal Chowdhary, Pushpam Virenbhai Patel, Krupal Jaysukbhai Kathrotia	The majority of imaging techniques use symmetric and asymmetric cryptography algorithms to encrypt digital media. Most of the research works contributed in the literature focus primarily on the Advanced Encryption Standard (AES) algorithm for encryption and decryption.	This helped us in getting a better idea on how to apply modern AES

10	Color image encryption and decryption using fractional Fourier transform Madhusuda Joshi, Chandrashakher, Kehar Singh	We propose the encryption of color images using fractional Fourier transform (FRT). The image to be encrypted is first segregated into three color channels: red, green, and blue. Each of these channels is encrypted independently using double random phase encoding in the FRT domain.	We learnt about the Fourier transform as a means of encryption and decryption
11	Image Encryption and Decryption Using SCAN Methodology Chao-shen Chen, Rong-jian Chen	This paper proposed an image encryption and decryption process. Its encryption method is based on SCAN patterns generated by the SCAN methodology	We learnt about the SCAN patterns and its methodology as a means of encryption and decryption
12	A new chaotic key-based design for image encryption and decryption Jui-Cheng Jiun-In Guo	In this paper, an image encryption/decryption algorithm and its VLSI architecture are proposed. According to a chaotic binary sequence, the gray level of each pixel is XORed or XNORed bit-by-bit to one of the two predetermined keys.	We learnt about the chaotic binary sequence as a means of encryption and decryption whilst getting an understanding about the VLSI architecture

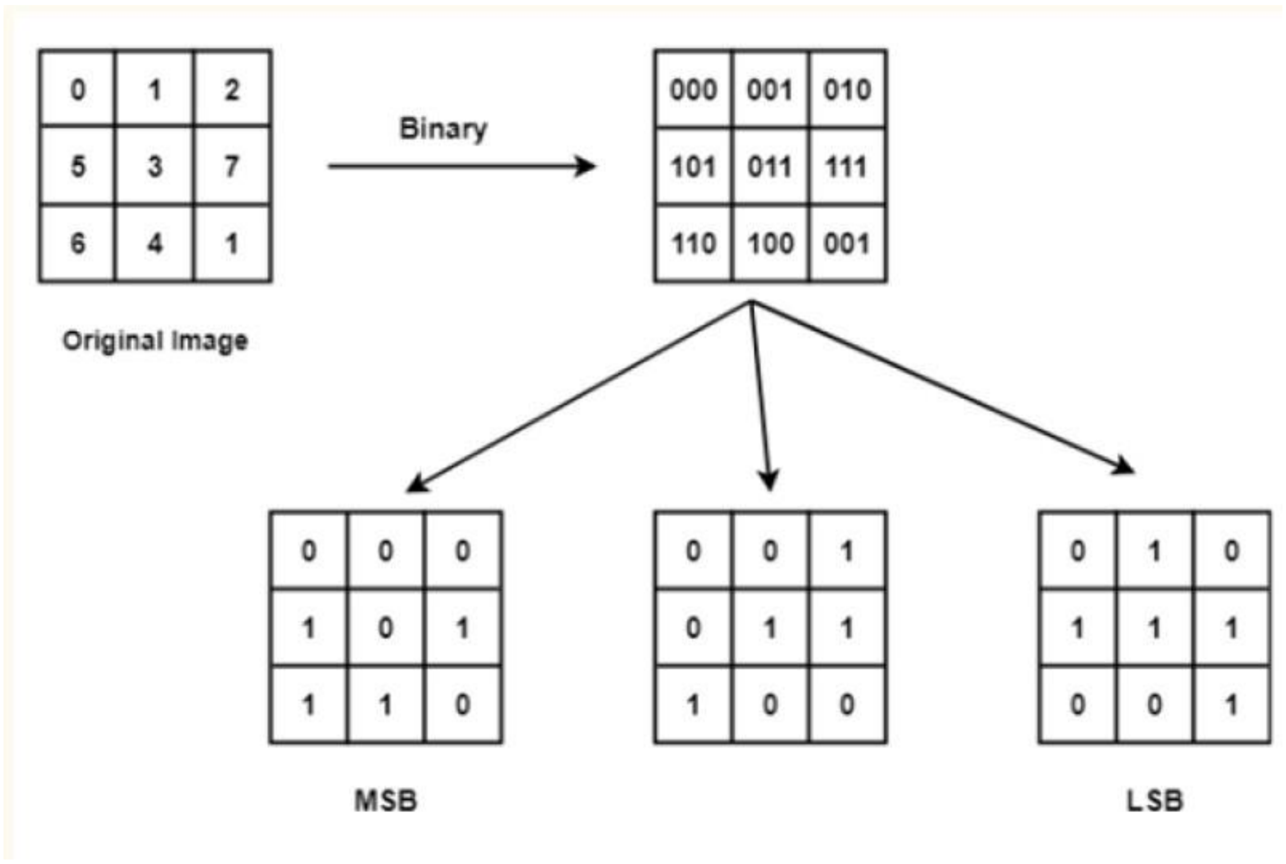
Design and Architecture



Bit Plane Slicing

One or more bits of the byte are used for each pixel when an image is represented via bit plane slicing. The original grey level is converted to a binary picture since only the MSB may be used to represent a pixel. Bit plane slicing has three primary objectives:

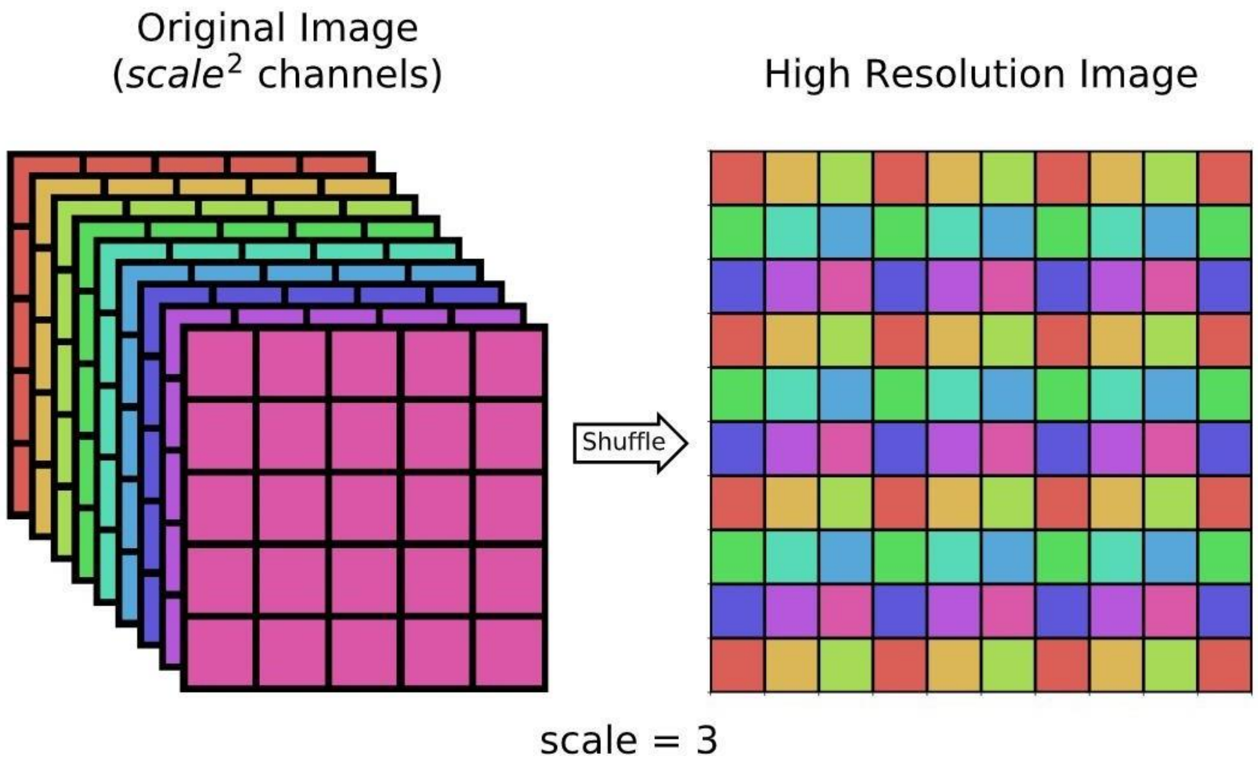
- Converting a gray level image to a binary image.
- Representing an image with fewer bits and corresponding the image to a smaller size.
- Enhancing the image by focusing



Transposition of pixels

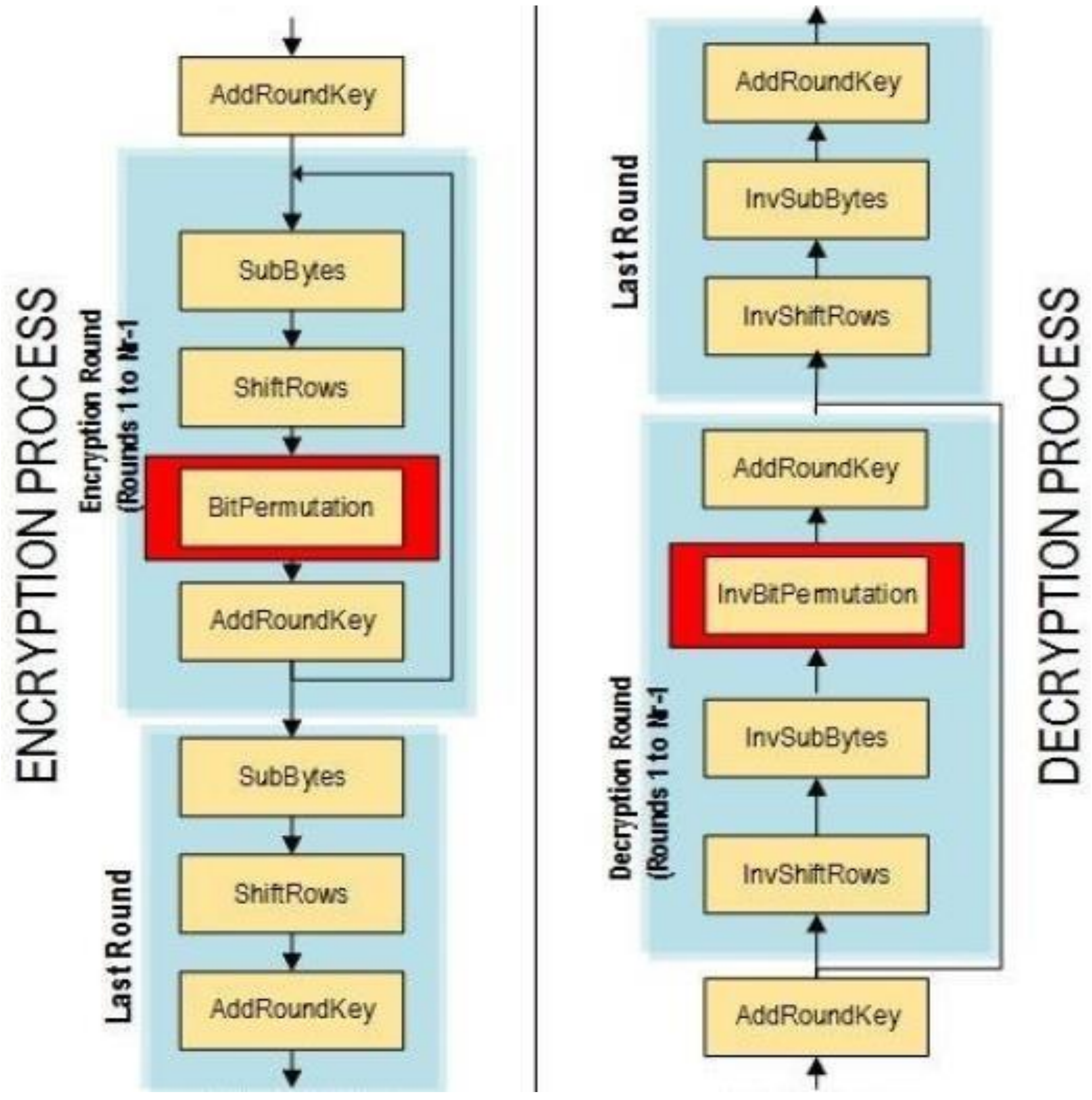
The method we used to shuffle the picture fragments produced after slicing is called Transposition of Pixels. This adds an additional layer, much like a jigsaw puzzle, to provide greater safety.

- Extracts the rgb values of the image
- Shuffles the pixel positions
- Adds a small border of width M to every pixel (it actually paints a line over existing pixels)
- Recreates an image with the shuffled and updated pixels
- Saves the recreated image

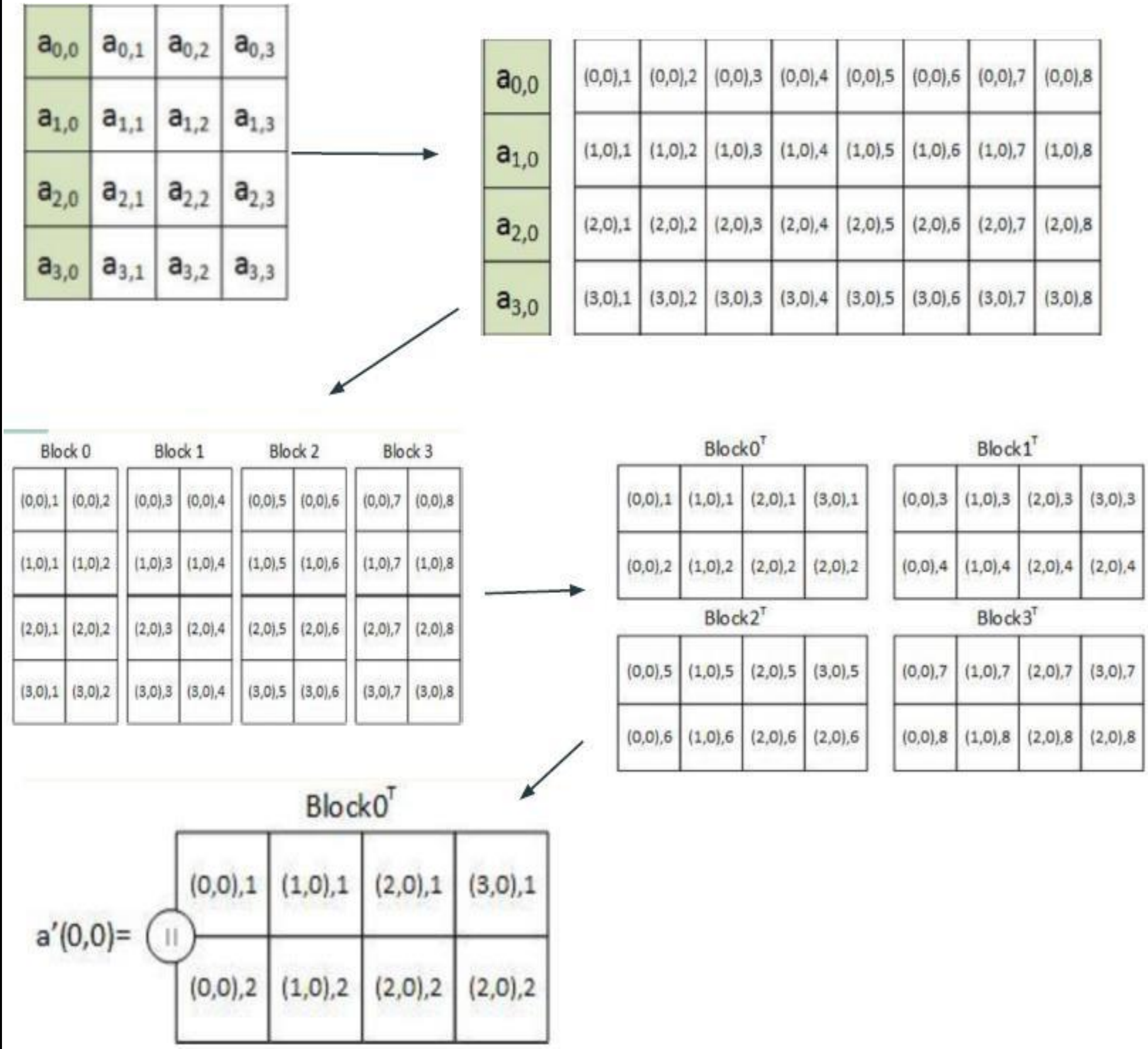


Modified AES(Bit Permutation)

Because of its effectiveness and simplicity, the AES encryption algorithm is one of the most widely utilised. AES employs straightforward algebraic approaches for encryption, yet it turns out to be insecure.
We suggest using permutation in instead of mix columns. A permutation strategy is used in an encryption algorithm to reduce the amount of memory needed, achieve a minimum encryption time, and make the algorithm simple to use.



Bit Permutation



$a_{0,0}$	$a_{0,1}$	$a_{0,2}$	$a_{0,3}$
$a_{1,0}$	$a_{1,1}$	$a_{1,2}$	$a_{1,3}$
$a_{2,0}$	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$
$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$

$a_{0,0}$	(0,0),1	(0,0),2	(0,0),3	(0,0),4	(0,0),5	(0,0),6	(0,0),7	(0,0),8
$a_{1,0}$	(1,0),1	(1,0),2	(1,0),3	(1,0),4	(1,0),5	(1,0),6	(1,0),7	(1,0),8
$a_{2,0}$	(2,0),1	(2,0),2	(2,0),3	(2,0),4	(2,0),5	(2,0),6	(2,0),7	(2,0),8
$a_{3,0}$	(3,0),1	(3,0),2	(3,0),3	(3,0),4	(3,0),5	(3,0),6	(3,0),7	(3,0),8

Block 0		Block 1		Block 2		Block 3	
(0,0),1	(0,0),2	(0,0),3	(0,0),4	(0,0),5	(0,0),6	(0,0),7	(0,0),8
(1,0),1	(1,0),2	(1,0),3	(1,0),4	(1,0),5	(1,0),6	(1,0),7	(1,0),8
(2,0),1	(2,0),2	(2,0),3	(2,0),4	(2,0),5	(2,0),6	(2,0),7	(2,0),8
(3,0),1	(3,0),2	(3,0),3	(3,0),4	(3,0),5	(3,0),6	(3,0),7	(3,0),8

Block0^T

(0,0),1	(1,0),1	(2,0),1	(3,0),1
(0,0),2	(1,0),2	(2,0),2	(2,0),2

Block1^T

(0,0),3	(1,0),3	(2,0),3	(3,0),3
(0,0),4	(1,0),4	(2,0),4	(2,0),4

Block2^T

(0,0),5	(1,0),5	(2,0),5	(3,0),5
(0,0),6	(1,0),6	(2,0),6	(2,0),6

Block3^T

(0,0),7	(1,0),7	(2,0),7	(3,0),7
(0,0),8	(1,0),8	(2,0),8	(2,0),8

Block0^T

a'(0,0)= $\parallel$

(0,0),1	(1,0),1	(2,0),1	(3,0),1
(0,0),2	(1,0),2	(2,0),2	(2,0),2

Code of our program

```
import numpy as np
import cv2

# Read the image in greyscale
img = cv2.imread('C:\\Users\\selva\\Downloads\\tree-736885__480.jpg', 0)

# Iterate over each pixel and change pixel value to binary using np.binary_repr() and
store it in a list.
lst = []
for i in range(img.shape[0]):
    for j in range(img.shape[1]):
        lst.append(np.binary_repr(img[i][j], width=8))  # width = no. of bits

# We have a list of strings where each string represents binary pixel value. To extract
bit planes we need to iterate over the strings and store the characters corresponding
to bit planes into lists.
# Multiply with 2^(n-1) and reshape to reconstruct the bit image.
eight_bit_img = (np.array([int(i[0]) for i in lst], dtype=np.uint8) *
128).reshape(img.shape[0], img.shape[1])
seven_bit_img = (np.array([int(i[1]) for i in lst], dtype=np.uint8) *
64).reshape(img.shape[0], img.shape[1])
six_bit_img = (np.array([int(i[2]) for i in lst], dtype=np.uint8) *
32).reshape(img.shape[0], img.shape[1])
five_bit_img = (np.array([int(i[3]) for i in lst], dtype=np.uint8) *
16).reshape(img.shape[0], img.shape[1])
four_bit_img = (np.array([int(i[4]) for i in lst], dtype=np.uint8) *
8).reshape(img.shape[0], img.shape[1])
three_bit_img = (np.array([int(i[5]) for i in lst], dtype=np.uint8) *
4).reshape(img.shape[0], img.shape[1])
two_bit_img = (np.array([int(i[6]) for i in lst], dtype=np.uint8) *
2).reshape(img.shape[0], img.shape[1])
one_bit_img = (np.array([int(i[7]) for i in lst], dtype=np.uint8) *
1).reshape(img.shape[0], img.shape[1])

# Concatenate these images for ease of display using cv2.hconcat()
finalr = cv2.hconcat([eight_bit_img, seven_bit_img, six_bit_img, five_bit_img])
finalv = cv2.hconcat([four_bit_img, three_bit_img, two_bit_img, one_bit_img])

# Vertically concatenate
final = cv2.vconcat([finalr, finalv])

# Display the images
cv2.imshow('a', final)
cv2.waitKey(0)

# Combining 4 bit planes #transposition of pixel(encryption)
new_img = eight_bit_img + seven_bit_img + six_bit_img + five_bit_img
# Display the image
cv2.imshow('Final image', new_img)
cv2.waitKey(0)

#transposition of pixel(encryption)
from PIL import Image

im = Image.open('image.png')
px = im.load()
w, h = im.size

y0 = 1
x0 = 1.0000001
vals = []
for i in range(0, h * w):
    x = 1 - 1.4 * pow(x0, 2) + y0
    y = 0.3 * x0
    xr = int(('%.11f' % (x))[4:9]) % w
    yr = int(('%.11f' % (y))[4:9]) % h
    vals.append((xr, yr))
    x0 = float('%.14f' % (x))
    y0 = float('%.14f' % (y))

vals.reverse()
for i in range(0, h * w):
    (xr, yr) = vals[i]
    j = h * w - i - 1
    p = px[j % w, int(j / w)]
    pr = px[xr, yr]
    px[j % w, int(j / w)] = pr
    px[xr, yr] = p

im.save('encrypted.png')

#transposition of pixels(decryption)
from PIL import Image

im = Image.open('encrypted.png')
px = im.load()
w, h = im.size

y0 = 1
x0 = 1.0000001
for i in range(0, h * w):
    x = 1 - 1.4 * pow(x0, 2) + y0
    y = 0.3 * x0
    x0 = float('%.14f' % (x))
    y0 = float('%.14f' % (y))
    xr = int(('%.11f' % (x))[4:9]) % w
    yr = int(('%.11f' % (y))[4:9]) % h

    p = px[i % w, int(i / w)]
    pr = px[xr, yr]
    px[i % w, int(i / w)] = pr
    px[xr, yr] = p

im.save('decrypted.jpg', optimize=False, progressive=False, quality=100)


#Modified AES
# Rijndael S-box
sbox = [0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x01, 0x67, 0x2b, 0xfe,
0xd7, 0xab, 0x76,
        0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0, 0xad, 0xd4, 0xa2, 0xaf, 0x9c,
0xa4, 0x72, 0xc0,
        0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc, 0x34, 0xa5, 0xe5, 0xf1, 0x71,
0xd8, 0x31, 0x15,
        0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a, 0x07, 0x12, 0x80, 0xe2, 0xeb,
0x27, 0xb2, 0x75,
        0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0, 0x52, 0x3b, 0xd6, 0xb3, 0x29,
0xe3, 0x2f, 0x84,
        0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b, 0x6a, 0xcb, 0xbe, 0x39, 0x4a,
0x4c, 0x58, 0xcf,
        0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85, 0x45, 0xf9, 0x02, 0x7f, 0x50,
0x3c, 0x9f, 0xa8,
        0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5, 0xbc, 0xb6, 0xda, 0x21, 0x10,
0xff, 0xf3, 0xd2,
        0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17, 0xc4, 0xa7, 0x7e, 0x3d, 0x64,
0x5d, 0x19, 0x73,
        0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88, 0x46, 0xee, 0xb8, 0x14, 0xde,
0x5e, 0x0b, 0xdb,
        0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c, 0xc2, 0xd3, 0xac, 0x62, 0x91,
0x95, 0xe4, 0x79,
    0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9, 0x6c, 0x56, 0xf4, 0xea, 0x65,
0x7a, 0xae, 0x08,
    0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6, 0xe8, 0xdd, 0x74, 0x1f, 0x4b,
0xbd, 0x8b, 0x8a,
    0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e, 0x61, 0x35, 0x57, 0xb9, 0x86,
0xc1, 0x1d, 0x9e,
    0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94, 0x9b, 0x1e, 0x87, 0xe9, 0xce,
0x55, 0x28, 0xdf,
    0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68, 0x41, 0x99, 0x2d, 0x0f, 0xb0,
0x54, 0xbb, 0x16]

# Rijndael Inverted S-box
rsbox = [0x52, 0x09, 0x6a, 0xd5, 0x30, 0x36, 0xa5, 0x38, 0xbf, 0x40, 0xa3, 0x9e, 0x81,
0xf3, 0xd7, 0xfb,
     0x7c, 0xe3, 0x39, 0x82, 0x9b, 0x2f, 0xff, 0x87, 0x34, 0x8e, 0x43, 0x44, 0xc4,
0xde, 0xe9, 0xcb,
     0x54, 0x7b, 0x94, 0x32, 0xa6, 0xc2, 0x23, 0x3d, 0xee, 0x4c, 0x95, 0x0b, 0x42,
0xfa, 0xc3, 0x4e,
     0x08, 0x2e, 0xa1, 0x66, 0x28, 0xd9, 0x24, 0xb2, 0x76, 0x5b, 0xa2, 0x49, 0x6d,
0x8b, 0xd1, 0x25,
     0x72, 0xf8, 0xf6, 0x64, 0x86, 0x68, 0x98, 0x16, 0xd4, 0xa4, 0x5c, 0xcc, 0x5d,
0x65, 0xb6, 0x92,
     0x6c, 0x70, 0x48, 0x50, 0xfd, 0xed, 0xb9, 0xda, 0x5e, 0x15, 0x46, 0x57, 0xa7,
0x8d, 0x9d, 0x84,
     0x90, 0xd8, 0xab, 0x00, 0x8c, 0xbc, 0xd3, 0x0a, 0xf7, 0xe4, 0x58, 0x05, 0xb8,
0xb3, 0x45, 0x06,
     0xd0, 0x2c, 0x1e, 0x8f, 0xca, 0x3f, 0x0f, 0x02, 0xc1, 0xaf, 0xbd, 0x03, 0x01,
0x13, 0x8a, 0x6b,
     0x3a, 0x91, 0x11, 0x41, 0x4f, 0x67, 0xdc, 0xea, 0x97, 0xf2, 0xcf, 0xce, 0xf0,
0xb4, 0xe6, 0x73,
     0x96, 0xac, 0x74, 0x22, 0xe7, 0xad, 0x35, 0x85, 0xe2, 0xf9, 0x37, 0xe8, 0x1c,
0x75, 0xdf, 0x6e,
     0x47, 0xf1, 0x1a, 0x71, 0x1d, 0x29, 0xc5, 0x89, 0x6f, 0xb7, 0x62, 0x0e, 0xaa,
0x18, 0xbe, 0x1b,
     0xfc, 0x56, 0x3e, 0x4b, 0xc6, 0xd2, 0x79, 0x20, 0x9a, 0xdb, 0xc0, 0xfe, 0x78,
0xcd, 0x5a, 0xf4,
     0x1f, 0xdd, 0xa8, 0x33, 0x88, 0x07, 0xc7, 0x31, 0xb1, 0x12, 0x10, 0x59, 0x27,
0x80, 0xec, 0x5f,
     0x60, 0x51, 0x7f, 0xa9, 0x19, 0xb5, 0x4a, 0x0d, 0x2d, 0xe5, 0x7a, 0x9f, 0x93,
0xc9, 0x9c, 0xef,
     0xa0, 0xe0, 0x3b, 0x4d, 0xae, 0x2a, 0xf5, 0xb0, 0xc8, 0xeb, 0xbb, 0x3c, 0x83,
0x53, 0x99, 0x61,
     0x17, 0x2b, 0x04, 0x7e, 0xba, 0x77, 0xd6, 0x26, 0xe1, 0x69, 0x14, 0x63, 0x55,
0x21, 0x0c, 0x7d]


mul2 = [0x00, 0x02, 0x04, 0x06, 0x08, 0x0a, 0x0c, 0x0e, 0x10, 0x12, 0x14, 0x16, 0x18,
0x1a, 0x1c, 0x1e,
    0x20, 0x22, 0x24, 0x26, 0x28, 0x2a, 0x2c, 0x2e, 0x30, 0x32, 0x34, 0x36, 0x38,
0x3a, 0x3c, 0x3e,
    0x40, 0x42, 0x44, 0x46, 0x48, 0x4a, 0x4c, 0x4e, 0x50, 0x52, 0x54, 0x56, 0x58,
0x5a, 0x5c, 0x5e,
    0x60, 0x62, 0x64, 0x66, 0x68, 0x6a, 0x6c, 0x6e, 0x70, 0x72, 0x74, 0x76, 0x78,
0x7a, 0x7c, 0x7e,
    0x80, 0x82, 0x84, 0x86, 0x88, 0x8a, 0x8c, 0x8e, 0x90, 0x92, 0x94, 0x96, 0x98,
0x9a, 0x9c, 0x9e,
    0xa0, 0xa2, 0xa4, 0xa6, 0xa8, 0xaa, 0xac, 0xae, 0xb0, 0xb2, 0xb4, 0xb6, 0xb8,
0xba, 0xbc, 0xbe,
    0xc0, 0xc2, 0xc4, 0xc6, 0xc8, 0xca, 0xcc, 0xce, 0xd0, 0xd2, 0xd4, 0xd6, 0xd8,
0xda, 0xdc, 0xde,
    0xe0, 0xe2, 0xe4, 0xe6, 0xe8, 0xea, 0xec, 0xee, 0xf0, 0xf2, 0xf4, 0xf6, 0xf8,
0xfa, 0xfc, 0xfe,
    0x1b, 0x19, 0x1f, 0x1d, 0x13, 0x11, 0x17, 0x15, 0x0b, 0x09, 0x0f, 0x0d, 0x03,
0x01, 0x07, 0x05,
    0x3b, 0x39, 0x3f, 0x3d, 0x33, 0x31, 0x37, 0x35, 0x2b, 0x29, 0x2f, 0x2d, 0x23,
0x21, 0x27, 0x25,
        0x5b, 0x59, 0x5f, 0x5d, 0x53, 0x51, 0x57, 0x55, 0x4b, 0x49, 0x4f, 0x4d, 0x43,
0x41, 0x47, 0x45,
        0x7b, 0x79, 0x7f, 0x7d, 0x73, 0x71, 0x77, 0x75, 0x6b, 0x69, 0x6f, 0x6d, 0x63,
0x61, 0x67, 0x65,
        0x9b, 0x99, 0x9f, 0x9d, 0x93, 0x91, 0x97, 0x95, 0x8b, 0x89, 0x8f, 0x8d, 0x83,
0x81, 0x87, 0x85,
        0xbb, 0xb9, 0xbf, 0xbd, 0xb3, 0xb1, 0xb7, 0xb5, 0xab, 0xa9, 0xaf, 0xad, 0xa3,
0xa1, 0xa7, 0xa5,
        0xdb, 0xd9, 0xdf, 0xdd, 0xd3, 0xd1, 0xd7, 0xd5, 0xcb, 0xc9, 0xcf, 0xcd, 0xc3,
0xc1, 0xc7, 0xc5,
        0xfb, 0xf9, 0xff, 0xfd, 0xf3, 0xf1, 0xf7, 0xf5, 0xeb, 0xe9, 0xef, 0xed, 0xe3,
0xe1, 0xe7, 0xe5]


mul3 = [0x00, 0x03, 0x06, 0x05, 0x0c, 0x0f, 0x0a, 0x09, 0x18, 0x1b, 0x1e, 0x1d, 0x14,
0x17, 0x12, 0x11,
        0x30, 0x33, 0x36, 0x35, 0x3c, 0x3f, 0x3a, 0x39, 0x28, 0x2b, 0x2e, 0x2d, 0x24,
0x27, 0x22, 0x21,
        0x60, 0x63, 0x66, 0x65, 0x6c, 0x6f, 0x6a, 0x69, 0x78, 0x7b, 0x7e, 0x7d, 0x74,
0x77, 0x72, 0x71,
        0x50, 0x53, 0x56, 0x55, 0x5c, 0x5f, 0x5a, 0x59, 0x48, 0x4b, 0x4e, 0x4d, 0x44,
0x47, 0x42, 0x41,
        0xc0, 0xc3, 0xc6, 0xc5, 0xcc, 0xcf, 0xca, 0xc9, 0xd8, 0xdb, 0xde, 0xdd, 0xd4,
0xd7, 0xd2, 0xd1,
        0xf0, 0xf3, 0xf6, 0xf5, 0xfc, 0xff, 0xfa, 0xf9, 0xe8, 0xeb, 0xee, 0xed, 0xe4,
0xe7, 0xe2, 0xe1,
        0xa0, 0xa3, 0xa6, 0xa5, 0xac, 0xaf, 0xaa, 0xa9, 0xb8, 0xbb, 0xbe, 0xbd, 0xb4,
0xb7, 0xb2, 0xb1,
        0x90, 0x93, 0x96, 0x95, 0x9c, 0x9f, 0x9a, 0x99, 0x88, 0x8b, 0x8e, 0x8d, 0x84,
0x87, 0x82, 0x81,
        0x9b, 0x98, 0x9d, 0x9e, 0x97, 0x94, 0x91, 0x92, 0x83, 0x80, 0x85, 0x86, 0x8f,
0x8c, 0x89, 0x8a,
        0xab, 0xa8, 0xad, 0xae, 0xa7, 0xa4, 0xa1, 0xa2, 0xb3, 0xb0, 0xb5, 0xb6, 0xbf,
0xbc, 0xb9, 0xba,
        0xfb, 0xf8, 0xfd, 0xfe, 0xf7, 0xf4, 0xf1, 0xf2, 0xe3, 0xe0, 0xe5, 0xe6, 0xef,
0xec, 0xe9, 0xea,
        0xcb, 0xc8, 0xcd, 0xce, 0xc7, 0xc4, 0xc1, 0xc2, 0xd3, 0xd0, 0xd5, 0xd6, 0xdf,
0xdc, 0xd9, 0xda,
        0x5b, 0x58, 0x5d, 0x5e, 0x57, 0x54, 0x51, 0x52, 0x43, 0x40, 0x45, 0x46, 0x4f,
0x4c, 0x49, 0x4a,
        0x6b, 0x68, 0x6d, 0x6e, 0x67, 0x64, 0x61, 0x62, 0x73, 0x70, 0x75, 0x76, 0x7f,
0x7c, 0x79, 0x7a,
        0x3b, 0x38, 0x3d, 0x3e, 0x37, 0x34, 0x31, 0x32, 0x23, 0x20, 0x25, 0x26, 0x2f,
0x2c, 0x29, 0x2a,
        0x0b, 0x08, 0x0d, 0x0e, 0x07, 0x04, 0x01, 0x02, 0x13, 0x10, 0x15, 0x16, 0x1f,
0x1c, 0x19, 0x1a]


Rcon = [0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36, 0x6c, 0xd8,
0xab, 0x4d, 0x9a,
        0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3, 0x7d, 0xfa, 0xef,
0xc5, 0x91, 0x39, ]

mul09 = [0x00, 0x09, 0x12, 0x1b, 0x24, 0x2d, 0x36, 0x3f, 0x48, 0x41, 0x5a, 0x53, 0x6c,
0x65, 0x7e, 0x77,
         0x90, 0x99, 0x82, 0x8b, 0xb4, 0xbd, 0xa6, 0xaf, 0xd8, 0xd1, 0xca, 0xc3, 0xfc,
0xf5, 0xee, 0xe7,
         0x3b, 0x32, 0x29, 0x20, 0x1f, 0x16, 0x0d, 0x04, 0x73, 0x7a, 0x61, 0x68, 0x57,
0x5e, 0x45, 0x4c,
         0xab, 0xa2, 0xb9, 0xb0, 0x8f, 0x86, 0x9d, 0x94, 0xe3, 0xea, 0xf1, 0xf8, 0xc7,
0xce, 0xd5, 0xdc,
         0x76, 0x7f, 0x64, 0x6d, 0x52, 0x5b, 0x40, 0x49, 0x3e, 0x37, 0x2c, 0x25, 0x1a,
0x13, 0x08, 0x01,
         0xe6, 0xef, 0xf4, 0xfd, 0xc2, 0xcb, 0xd0, 0xd9, 0xae, 0xa7, 0xbc, 0xb5, 0x8a,
0x83, 0x98, 0x91,
         0x4d, 0x44, 0x5f, 0x56, 0x69, 0x60, 0x7b, 0x72, 0x05, 0x0c, 0x17, 0x1e, 0x21,
0x28, 0x33, 0x3a,
          0xdd, 0xd4, 0xcf, 0xc6, 0xf9, 0xf0, 0xeb, 0xe2, 0x95, 0x9c, 0x87, 0x8e, 0xb1,
0xb8, 0xa3, 0xaa,
          0xec, 0xe5, 0xfe, 0xf7, 0xc8, 0xc1, 0xda, 0xd3, 0xa4, 0xad, 0xb6, 0xbf, 0x80,
0x89, 0x92, 0x9b,
          0x7c, 0x75, 0x6e, 0x67, 0x58, 0x51, 0x4a, 0x43, 0x34, 0x3d, 0x26, 0x2f, 0x10,
0x19, 0x02, 0x0b,
          0xd7, 0xde, 0xc5, 0xcc, 0xf3, 0xfa, 0xe1, 0xe8, 0x9f, 0x96, 0x8d, 0x84, 0xbb,
0xb2, 0xa9, 0xa0,
          0x47, 0x4e, 0x55, 0x5c, 0x63, 0x6a, 0x71, 0x78, 0x0f, 0x06, 0x1d, 0x14, 0x2b,
0x22, 0x39, 0x30,
          0x9a, 0x93, 0x88, 0x81, 0xbe, 0xb7, 0xac, 0xa5, 0xd2, 0xdb, 0xc0, 0xc9, 0xf6,
0xff, 0xe4, 0xed,
          0x0a, 0x03, 0x18, 0x11, 0x2e, 0x27, 0x3c, 0x35, 0x42, 0x4b, 0x50, 0x59, 0x66,
0x6f, 0x74, 0x7d,
          0xa1, 0xa8, 0xb3, 0xba, 0x85, 0x8c, 0x97, 0x9e, 0xe9, 0xe0, 0xfb, 0xf2, 0xcd,
0xc4, 0xdf, 0xd6,
          0x31, 0x38, 0x23, 0x2a, 0x15, 0x1c, 0x07, 0x0e, 0x79, 0x70, 0x6b, 0x62, 0x5d,
0x54, 0x4f, 0x46]

mul0b = [0x00, 0x0b, 0x16, 0x1d, 0x2c, 0x27, 0x3a, 0x31, 0x58, 0x53, 0x4e, 0x45, 0x74,
0x7f, 0x62, 0x69,
          0xb0, 0xbb, 0xa6, 0xad, 0x9c, 0x97, 0x8a, 0x81, 0xe8, 0xe3, 0xfe, 0xf5, 0xc4,
0xcf, 0xd2, 0xd9,
          0x7b, 0x70, 0x6d, 0x66, 0x57, 0x5c, 0x41, 0x4a, 0x23, 0x28, 0x35, 0x3e, 0x0f,
0x04, 0x19, 0x12,
          0xcb, 0xc0, 0xdd, 0xd6, 0xe7, 0xec, 0xf1, 0xfa, 0x93, 0x98, 0x85, 0x8e, 0xbf,
0xb4, 0xa9, 0xa2,
          0xf6, 0xfd, 0xe0, 0xeb, 0xda, 0xd1, 0xcc, 0xc7, 0xae, 0xa5, 0xb8, 0xb3, 0x82,
0x89, 0x94, 0x9f,
          0x46, 0x4d, 0x50, 0x5b, 0x6a, 0x61, 0x7c, 0x77, 0x1e, 0x15, 0x08, 0x03, 0x32,
0x39, 0x24, 0x2f,
          0x8d, 0x86, 0x9b, 0x90, 0xa1, 0xaa, 0xb7, 0xbc, 0xd5, 0xde, 0xc3, 0xc8, 0xf9,
0xf2, 0xef, 0xe4,
          0x3d, 0x36, 0x2b, 0x20, 0x11, 0x1a, 0x07, 0x0c, 0x65, 0x6e, 0x73, 0x78, 0x49,
0x42, 0x5f, 0x54,
          0xf7, 0xfc, 0xe1, 0xea, 0xdb, 0xd0, 0xcd, 0xc6, 0xaf, 0xa4, 0xb9, 0xb2, 0x83,
0x88, 0x95, 0x9e,
          0x47, 0x4c, 0x51, 0x5a, 0x6b, 0x60, 0x7d, 0x76, 0x1f, 0x14, 0x09, 0x02, 0x33,
0x38, 0x25, 0x2e,
          0x8c, 0x87, 0x9a, 0x91, 0xa0, 0xab, 0xb6, 0xbd, 0xd4, 0xdf, 0xc2, 0xc9, 0xf8,
0xf3, 0xee, 0xe5,
          0x3c, 0x37, 0x2a, 0x21, 0x10, 0x1b, 0x06, 0x0d, 0x64, 0x6f, 0x72, 0x79, 0x48,
0x43, 0x5e, 0x55,
          0x01, 0x0a, 0x17, 0x1c, 0x2d, 0x26, 0x3b, 0x30, 0x59, 0x52, 0x4f, 0x44, 0x75,
0x7e, 0x63, 0x68,
          0xb1, 0xba, 0xa7, 0xac, 0x9d, 0x96, 0x8b, 0x80, 0xe9, 0xe2, 0xff, 0xf4, 0xc5,
0xce, 0xd3, 0xd8,
          0x7a, 0x71, 0x6c, 0x67, 0x56, 0x5d, 0x40, 0x4b, 0x22, 0x29, 0x34, 0x3f, 0x0e,
0x05, 0x18, 0x13,
          0xca, 0xc1, 0xdc, 0xd7, 0xe6, 0xed, 0xf0, 0xfb, 0x92, 0x99, 0x84, 0x8f, 0xbe,
0xb5, 0xa8, 0xa3]

mul0d = [0x00, 0x0d, 0x1a, 0x17, 0x34, 0x39, 0x2e, 0x23, 0x68, 0x65, 0x72, 0x7f, 0x5c,
0x51, 0x46, 0x4b,
          0xd0, 0xdd, 0xca, 0xc7, 0xe4, 0xe9, 0xfe, 0xf3, 0xb8, 0xb5, 0xa2, 0xaf, 0x8c,
0x81, 0x96, 0x9b,
          0xbb, 0xb6, 0xa1, 0xac, 0x8f, 0x82, 0x95, 0x98, 0xd3, 0xde, 0xc9, 0xc4, 0xe7,
0xea, 0xfd, 0xf0,
          0x6b, 0x66, 0x71, 0x7c, 0x5f, 0x52, 0x45, 0x48, 0x03, 0x0e, 0x19, 0x14, 0x37,
0x3a, 0x2d, 0x20,
          0x6d, 0x60, 0x77, 0x7a, 0x59, 0x54, 0x43, 0x4e, 0x05, 0x08, 0x1f, 0x12, 0x31,
0x3c, 0x2b, 0x26,
          0xbd, 0xb0, 0xa7, 0xaa, 0x89, 0x84, 0x93, 0x9e, 0xd5, 0xd8, 0xcf, 0xc2, 0xe1,
0xec, 0xfb, 0xf6,
          0xd6, 0xdb, 0xcc, 0xc1, 0xe2, 0xef, 0xf8, 0xf5, 0xbe, 0xb3, 0xa4, 0xa9, 0x8a,
0x87, 0x90, 0x9d,
          0x06, 0x0b, 0x1c, 0x11, 0x32, 0x3f, 0x28, 0x25, 0x6e, 0x63, 0x74, 0x79, 0x5a,
0x57, 0x40, 0x4d,
          0xda, 0xd7, 0xc0, 0xcd, 0xee, 0xe3, 0xf4, 0xf9, 0xb2, 0xbf, 0xa8, 0xa5, 0x86,
0x8b, 0x9c, 0x91,
          0x0a, 0x07, 0x10, 0x1d, 0x3e, 0x33, 0x24, 0x29, 0x62, 0x6f, 0x78, 0x75, 0x56,
0x5b, 0x4c, 0x41,
          0x61, 0x6c, 0x7b, 0x76, 0x55, 0x58, 0x4f, 0x42, 0x09, 0x04, 0x13, 0x1e, 0x3d,
0x30, 0x27, 0x2a,
          0xb1, 0xbc, 0xab, 0xa6, 0x85, 0x88, 0x9f, 0x92, 0xd9, 0xd4, 0xc3, 0xce, 0xed,
0xe0, 0xf7, 0xfa,
          0xb7, 0xba, 0xad, 0xa0, 0x83, 0x8e, 0x99, 0x94, 0xdf, 0xd2, 0xc5, 0xc8, 0xeb,
0xe6, 0xf1, 0xfc,
          0x67, 0x6a, 0x7d, 0x70, 0x53, 0x5e, 0x49, 0x44, 0x0f, 0x02, 0x15, 0x18, 0x3b,
0x36, 0x21, 0x2c,
          0x0c, 0x01, 0x16, 0x1b, 0x38, 0x35, 0x22, 0x2f, 0x64, 0x69, 0x7e, 0x73, 0x50,
0x5d, 0x4a, 0x47,
          0xdc, 0xd1, 0xc6, 0xcb, 0xe8, 0xe5, 0xf2, 0xff, 0xb4, 0xb9, 0xae, 0xa3, 0x80,
0x8d, 0x9a, 0x97]

mul0e = [0x00, 0x0e, 0x1c, 0x12, 0x38, 0x36, 0x24, 0x2a, 0x70, 0x7e, 0x6c, 0x62, 0x48,
0x46, 0x54, 0x5a,
          0xe0, 0xee, 0xfc, 0xf2, 0xd8, 0xd6, 0xc4, 0xca, 0x90, 0x9e, 0x8c, 0x82, 0xa8,
0xa6, 0xb4, 0xba,
          0xdb, 0xd5, 0xc7, 0xc9, 0xe3, 0xed, 0xff, 0xf1, 0xab, 0xa5, 0xb7, 0xb9, 0x93,
0x9d, 0x8f, 0x81,
          0x3b, 0x35, 0x27, 0x29, 0x03, 0x0d, 0x1f, 0x11, 0x4b, 0x45, 0x57, 0x59, 0x73,
0x7d, 0x6f, 0x61,
          0xad, 0xa3, 0xb1, 0xbf, 0x95, 0x9b, 0x89, 0x87, 0xdd, 0xd3, 0xc1, 0xcf, 0xe5,
0xeb, 0xf9, 0xf7,
          0x4d, 0x43, 0x51, 0x5f, 0x75, 0x7b, 0x69, 0x67, 0x3d, 0x33, 0x21, 0x2f, 0x05,
0x0b, 0x19, 0x17,
          0x76, 0x78, 0x6a, 0x64, 0x4e, 0x40, 0x52, 0x5c, 0x06, 0x08, 0x1a, 0x14, 0x3e,
0x30, 0x22, 0x2c,
          0x96, 0x98, 0x8a, 0x84, 0xae, 0xa0, 0xb2, 0xbc, 0xe6, 0xe8, 0xfa, 0xf4, 0xde,
0xd0, 0xc2, 0xcc,
          0x41, 0x4f, 0x5d, 0x53, 0x79, 0x77, 0x65, 0x6b, 0x31, 0x3f, 0x2d, 0x23, 0x09,
0x07, 0x15, 0x1b,
          0xa1, 0xaf, 0xbd, 0xb3, 0x99, 0x97, 0x85, 0x8b, 0xd1, 0xdf, 0xcd, 0xc3, 0xe9,
0xe7, 0xf5, 0xfb,
          0x9a, 0x94, 0x86, 0x88, 0xa2, 0xac, 0xbe, 0xb0, 0xea, 0xe4, 0xf6, 0xf8, 0xd2,
0xdc, 0xce, 0xc0,
          0x7a, 0x74, 0x66, 0x68, 0x42, 0x4c, 0x5e, 0x50, 0x0a, 0x04, 0x16, 0x18, 0x32,
0x3c, 0x2e, 0x20,
          0xec, 0xe2, 0xf0, 0xfe, 0xd4, 0xda, 0xc8, 0xc6, 0x9c, 0x92, 0x80, 0x8e, 0xa4,
0xaa, 0xb8, 0xb6,
          0x0c, 0x02, 0x10, 0x1e, 0x34, 0x3a, 0x28, 0x26, 0x7c, 0x72, 0x60, 0x6e, 0x44,
0x4a, 0x58, 0x56,
          0x37, 0x39, 0x2b, 0x25, 0x0f, 0x01, 0x13, 0x1d, 0x47, 0x49, 0x5b, 0x55, 0x7f,
0x71, 0x63, 0x6d,
          0xd7, 0xd9, 0xcb, 0xc5, 0xef, 0xe1, 0xf3, 0xfd, 0xa7, 0xa9, 0xbb, 0xb5, 0x9f,
0x91, 0x83, 0x8d]


def key_expansion_core(inn, i):
    # Rotate left
    t = inn[0]
    inn[0], inn[1], inn[2], inn[3] = inn[1], inn[2], inn[3], t
    # s_box four bytes
    inn[0], inn[1], inn[2], inn[3] = sbox[inn[0]], sbox[inn[1]], sbox[inn[2]],
sbox[inn[3]]
    # Rcon
    inn[0] ^= Rcon[i]
    return inn


def key_expansion(input_key):
    # the first 16 bytes are the original key
    expand_key = input_key[:16]
    expand_key += "0" * 160
    bytes_generated = 16
    rcon_iteration = 1
    # expanding keys to 11*16 bytes (4*11 words)
    while(bytes_generated < 176):
        # read 4 bytes for the key
        temp = []
        for i in range(4):
            temp.append(expand_key[i + bytes_generated - 4])
        # perform the core once for each 16 bytes key
        if(bytes_generated % 16 == 0):
            temp = key_expansion_core(temp, rcon_iteration)
            rcon_iteration += 1
        # XOR temp with [bytes_generated-16], and store in expanded_keys
        for a in range(4):
            expand_key[bytes_generated] = expand_key[bytes_generated-16] ^ temp[a]
            bytes_generated += 1

    return expand_key


def sub_byte(state):
    for i in range(16):
        state[i] = sbox[state[i]]
    return state


def inv_sub_byte(state):
    for i in range(16):
        state[i] = rsbox[state[i]]
    return state


def shift_rows(state):
    tmp = []
    tmp.append(state[0])
    tmp.append(state[5])
    tmp.append(state[10])
    tmp.append(state[15])
    tmp.append(state[4])
    tmp.append(state[9])
    tmp.append(state[14])
    tmp.append(state[3])
    tmp.append(state[8])
    tmp.append(state[13])
    tmp.append(state[2])
    tmp.append(state[7])
    tmp.append(state[12])
    tmp.append(state[1])
    tmp.append(state[6])
    tmp.append(state[11])

    return tmp


def inv_shift_rows(state):
    tmp = []
    tmp.append(state[0])
    tmp.append(state[13])
    tmp.append(state[10])
    tmp.append(state[7])
    tmp.append(state[4])
    tmp.append(state[1])
    tmp.append(state[14])
    tmp.append(state[11])
    tmp.append(state[8])
    tmp.append(state[5])
    tmp.append(state[2])
    tmp.append(state[15])
    tmp.append(state[12])
    tmp.append(state[9])
    tmp.append(state[6])
    tmp.append(state[3])
    return tmp


def int_to_8bit(a):
    bnr = bin(a).replace('0b', '')
    x = bnr[::-1] # this reverses an array
    while len(x) < 8:
        x += '0'
    bnr = x[::-1]
    return bnr


def add2(a):
    decimal = 0
    for digit in a:
        decimal = decimal * 2 + int(digit)
    return decimal


def listToInt(integers):
    strings = [str(integer) for integer in integers]
    a_string = "".join(strings)
    an_integer = int(a_string)
    return an_integer


def chunks(l, n):
    final = [l[i * n:(i + 1) * n] for i in range((len(l) + n - 1) // n)]
    return final


def Single_bit_permutation(a):
    rows = []
    for i in range(4):
        a_list = [int(x) for x in str(int_to_8bit(a[i]))]
        rows.append(a_list)
    list = []
    for i in range(8):
        for j in range(4):
            list.append((rows[j][i]))
    k = chunks(list, 8)

    for i in range(4):
        a = listToInt(k[i])
        k[i] = add2(str(a))
    return k

def bit_permutations(state):
    tmp = []

    for i in range(0, 16, 4):
        l = []
        for j in range(0, 4):
            l.append(state[i + j])
        l = Single_bit_permutation(l)
        for j in range(0, 4):
            tmp.append(l[j])

    return tmp
```

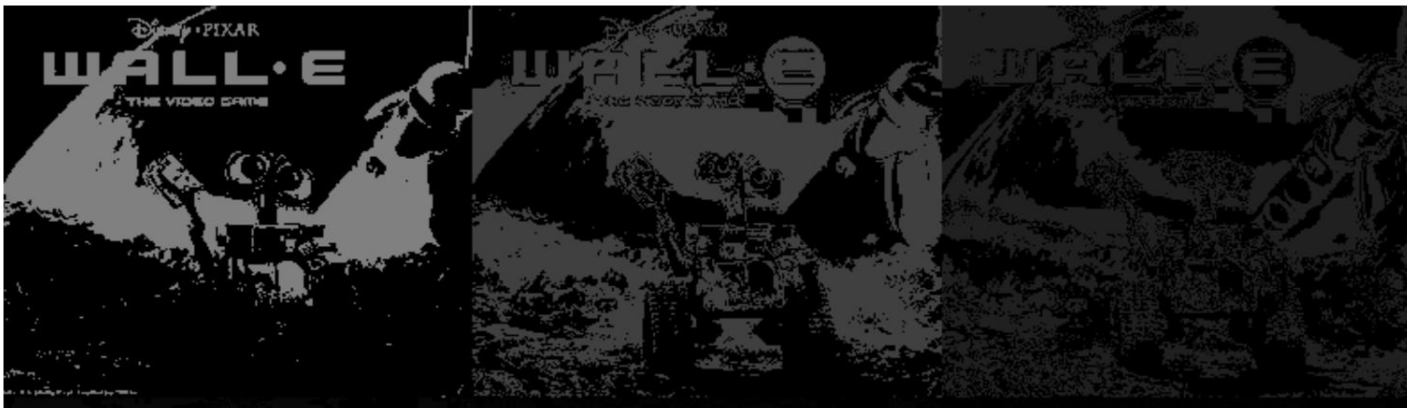
```python
def inv_Single_bit_permutation(a):
    rows = []
    for i in range(4):
        a_list = [int(x) for x in str(int_to_8bit(a[i]))]
        rows.append(a_list)
    list = []
    for i in range(4):
        for k in range(4):
            list.append(rows[k][i])
            list.append(rows[k][(i + 4) % 8])

    k = chunks(list, 8)
    for i in range(4):
        a = listToInt(k[i])
        k[i] = add2(str(a))
    return k


def inv_bit_permutations(state):
    tmp = []

    for i in range(0, 16, 4):
        l = []
        for j in range(0, 4):
            l.append(state[i + j])
        l = inv_Single_bit_permutation(l)
        for j in range(0, 4):
            tmp.append(l[j])

    return (tmp)

def add_round_key(state, round_key):
    for i in range(16):
        state[i] ^= round_key[i]

    return state


def aes_encrypt(msg, key):
    state = []
    for i in range(16):
        state.append((msg[i]))              #   ord() deleted
    number_of_rounds = 9
    expanded_keys = key_expansion(key)
    state = add_round_key(state, key)

    for round in range(number_of_rounds): # apply round operations
        state = sub_byte(state)
        state = shift_rows(state)
        state = bit_permutations(state)
        state = add_round_key(state, expanded_keys[(16*(round+1)):])

    # final round
    state = sub_byte(state)
    state = shift_rows(state)
    state = add_round_key(state, expanded_keys[160:])
    return state


def aes_decrypt(msg, key):
    state = []
    for i in range(16):
        state.append((msg[i]))
    number_of_rounds = 9
    expanded_keys = key_expansion(key)
    state = add_round_key(state, expanded_keys[160:])
    for round in range(number_of_rounds-1, -1, -1): # apply round operations
        state = inv_shift_rows(state)
        state = inv_sub_byte(state)
        state = add_round_key(state, expanded_keys[(16*(round+1)):])
        state = inv_bit_permutations(state)

    # final round
    state = inv_shift_rows(state)
    state = inv_sub_byte(state)
    state = add_round_key(state, key)
    return state


def image_encryption():
    path = "images.jpg"
    #key = key = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]
    print("\nEnter key: ")
    key = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]
    # open file for reading purpose
    fin = open(path, 'rb')
    # storing image data in variable "image"
    image = fin.read()
    fin.close()
    image = bytearray(image)
    img_len = len(image)
    print(img_len)
    padded_img_len = img_len
    if (img_len % 16 != 0):
        padded_img_len = (int)(img_len/16 + 1) * 16
    # add padding to byte array
    for i in range(padded_img_len - img_len):
        image += bytes([0])

    encrypted_img = []
    for i in range(0, padded_img_len, 16):
        encrypted_img += aes_encrypt(image[i:], key)
    encrypted_img = bytearray(encrypted_img)
    # opening file for writting purpose
    out_path = "encrypted_"+path
    fin = open(out_path, 'wb')
    # writing encrypted data in image
    fin.write(encrypted_img)
    fin.close()
    print('Encryption Done...')
    return


def image_decryption():
    path = "encrypted_images.jpg"
    #key = key = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]
    key = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16]
    # open file for reading purpose
    fin = open(path, 'rb')
    # storing image data in variable "image"
    image = fin.read()
    fin.close()
    image = bytearray(image)
    img_len = len(image)
    padded_img_len = img_len
    if (img_len % 16 != 0):
        padded_img_len = (int)(img_len/16 + 1) * 16
    # add padding to byte array
    for i in range(padded_img_len - img_len):
        image += bytes([0])

    decrypted_img = []
    for i in range(0, padded_img_len, 16):
        decrypted_img += aes_decrypt(image[i:], key)

    i = padded_img_len-1
    j = 0
    while(decrypted_img[i]== 0):
        i-=1
        j+=1

    decrypted_img = decrypted_img[:padded_img_len-j]
    decrypted_img = bytearray(decrypted_img)
    # opening file for writting purpose
    out_path = "decrypted_"+path
    fin = open(out_path, 'wb')
    # writing encrypted data in image
    fin.write(decrypted_img)
    fin.close()
    print('Decryption Done...')
    return




# ---------------               Main Function                 -----------------------
# ---------------                Starts here                  -----------------------


number2 = input("\nEnter 2 for image Encryption, 3 for Decryption: ")
if number2 == '2':
        image_encryption()
elif number2 == '3':
        image_decryption()
```

Implementation

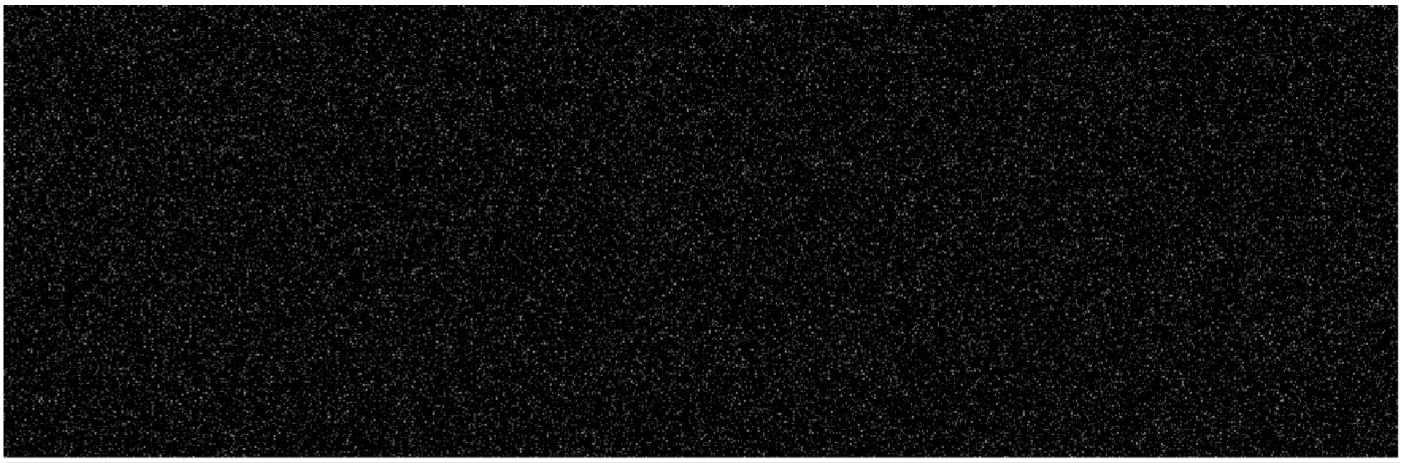
Input Image :

Disney • PIXAR
WALL•E
THE VIDEO GAME

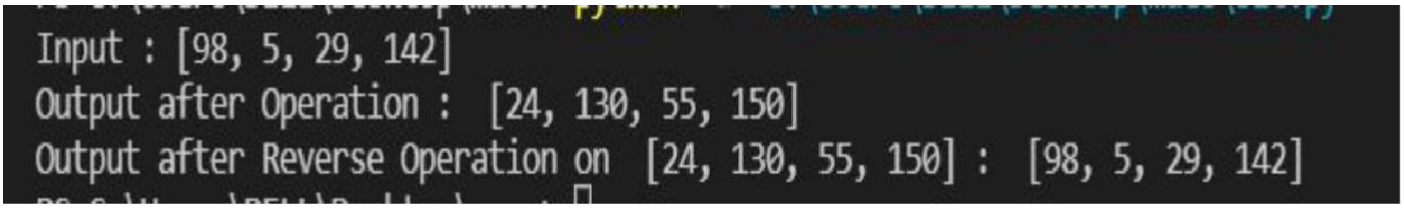
Sliced Image :



Transposition of pixels:



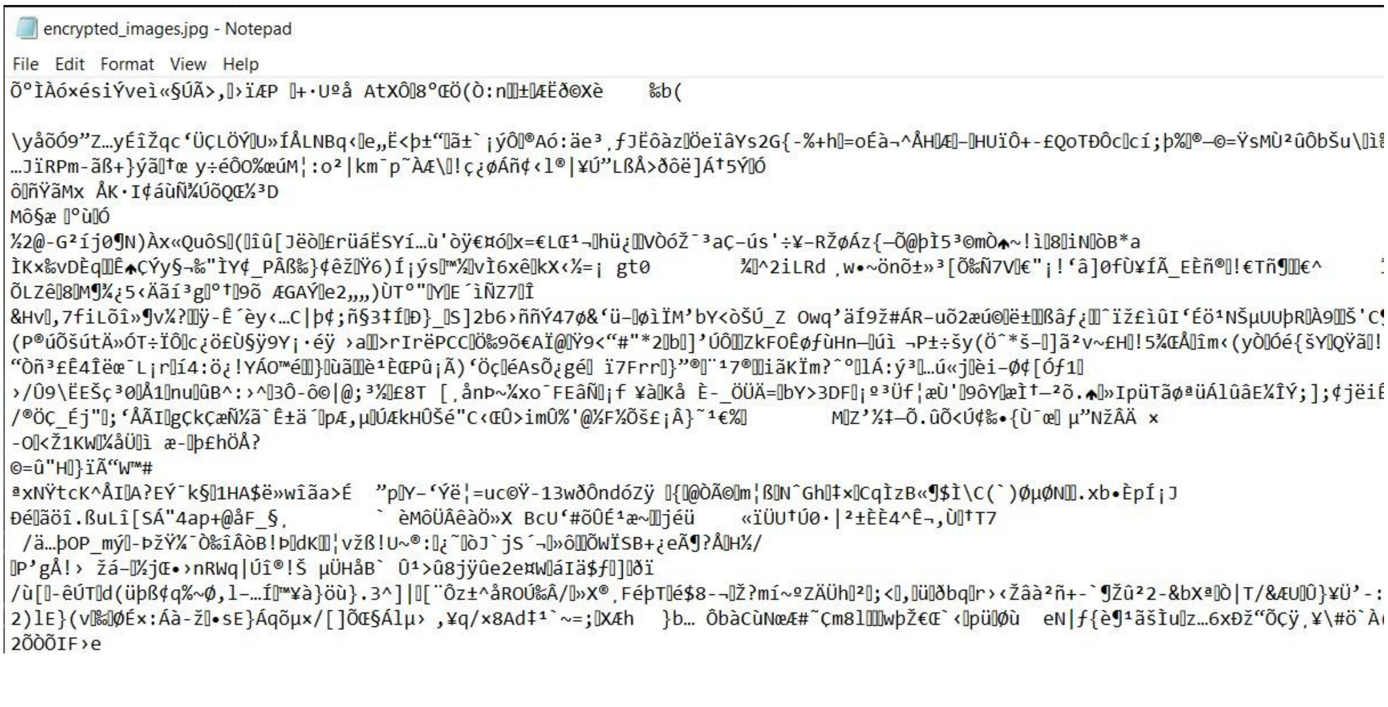
Bit permutation:



```
Input : [98, 5, 29, 142]
Output after Operation :  [24, 130, 55, 150]
Output after Reverse Operation on  [24, 130, 55, 150] :  [98, 5, 29, 142]
```

Modified Aes:

This results in cipher text and has to be opened in notepad



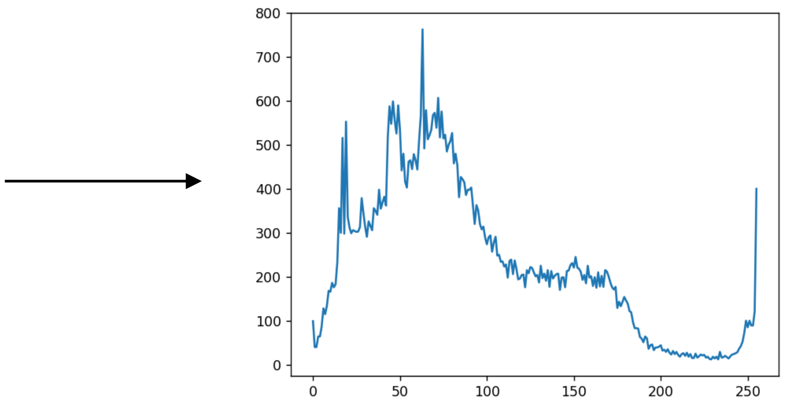
```
Õ°ÌÀó×ésiÝvei«§ÚÃ>,›ïÆP +·Uºå AtXÔ8°ŒÖ(Ò:n±ÆËð©Xè    ‰b(

\yåõó9”Z…yÉîŽqc‘ÜÇLÖÝU»ÍÅLNBq‹e„Ë<þ±“ã±`¡ýÔ®Aó:äe³¸ƒJËôàzÖeïâYs2G{-%+h=oÉà¬^ÅHÆ–HUïÔ+-£QoTÐÔccí;þ%®–©=ŸsMÙ²ûÔbŠu\ì
…JïRPm-ãß+}ýã†œ y÷éÔO%œúM¦:o²|km¯p˜ÀÆ\!ç¿øÁñ¢‹l®|¥Ú”LßÅ>ðöë]Á†5ÝÓ
ôñŸãMx ÅK·I¢áùÑ¼ÚõQŒ½³D
Mô§æ °ùÓ
½2@-G²íj0¶N)Àx«QuôS(îû[Jëò£rüáËSYí…ù'òÿ€¤óx=€LŒ¹¬hü¿VÒóŽ¯³aÇ-ús'÷¥-RŽøÁz{—Õ@þÌ5³©mÒ~!ì8iNòB*a
ÌK×‰vDÈqÊÇÝy§¬‰"ÌY¢_PÂß‰}¢êžŸ6)Í¡ýs™½vÌ6xêkX‹½=¡ gt0        ½^2iLRd¸w•~önõ±»³[Õ‰Ñ7V€"¡!‘â]0fÙ¥ÍÃ_EÈñ®!€Tñ¶€^
ÕLZê8M¶¾¿5‹Äãí³g°†9õ ÆGAÝe2„„)ÙT°"YE´ìÑZ7Î
&Hv,7fiLõî»¶v¾?ÿ-Ê´èy‹…C|þ¢;ñ§3‡ÍÐ}_S]2b6›ññÝ47ø&‘ü-øìÏM’bY<òŠÚ_Z Owq’äÍ9ž#ÁR-uõ2æú©ë±ßâƒ¿ˆïž£ìûI‘Éö¹NŠµUUþRÀ9Š'C
(P®úÕšútÄ»ÓT÷ÏÔc¿ö£Ù§ÿ9Y¦·éÿ ›a>rIrëPCCÖ‰9õ€AÏ@Ÿ9<“#"*2b]’ÚÔZkFOÊøƒùHn—úì ¬P±÷šy(Õˆ*š-]ã²v~£H!5¾ŒÅîm‹(yòÓé{šYQŸã!
“Õñ³£Ê4Îëœ¯L¡rí4:ö¿!YÁO™é}ùãè¹ÈŒPû¡Ã)‘ÖçéAsÕ¿gé ï7Frr}”®¨¹7®iãKÏm?ˆ°lÁ:ý³…ú«jèi-ø¢[Óƒ1
>/Û9\ËEŠç³0Å1nuûB^:›^3Ô-ô©|@;³½£8T [¸ånÞ~¾xo¯FEâÑ¡f ¥àKå È-_ÖÜÄ=bY>3DF¡º³Üf¦æÙ'9ôYæÌ†–²õ.»IpüTãøªüÁlûãE¾ÎÝ;];¢jëi
/®ÖÇ_Éj"‘ÅÃIgÇkÇæÑ½ã`Ê±ä´pÆ,µÚÆkHÛŠé"C‹ŒÛ>imÛ%'@½F½Õš£¡Â}˜¹€%        MZ’½‡–Õ.ûÕ<Ú¢‰•{Ù¯œ µ”NžÂÄ ×
-O<Ž1KW¼åÜì æ-þ£hÖÅ?
©=û"H}ïÃ“W™#
ªxNÝtcK^ÅIA?EÝ¯k§1HA$ë»wîãa>É  ”pY-‘Ýë¦=uc©Ÿ-13wðÔndóZÿ {@ÒÃ©m¦ßN^GhI‡×CqÌzB«¶$Ì\C(`)ØµØN.xb•Èpí¡J
Ðéäöî.ßuLî[SÁ"4ap+@åF_§¸        ` èMôÜÂêàÖ»X BcU‘#õÛÉ¹æ~jéü    «ïÜU†Ú0·|²±ÈÈ4^Ê¬,Ù†T7
 /ä…þOP_mý-ÞžŸ¼¯Ò‰îÂòB!ÞdK¦vžß!U~®:¿˜òJ`jS´¬»ôÕWÏSB+¿eÃ¶?ÅH½/
P’gÅ!› žá-½jŒ•›nRWq|Úî®!Š µÜHåB` Û¹›û8jÿûe2e¤WáIä$ƒ]ðï
/ù[-êÚTd(üþß¢q%~ø,l-…Í™¥à}öù}.3^]|[¨Ôz±^åROÚ‰Ã/»X®¸FéþTé$8-¬Ž?mí~ºZÄÜh²;<,üðbqr›‹Žâà²ñ+-`¶Žû²2-&bXªÒ|T/&ÆUÛ}¥Ü’-:
2)lE}(v‰ØÉ×:Áà-ž•sE}Áqõµ×/[]ÕŒ§Álµ› ,¥q/×8Ad‡¹`~=;XÆh  }b… ÔbàCùNœÆ#˜Çm8lwþŽ€Œ`‹püØù  eN|ƒ{è¶¹ãšÌuz…6xÐž“ÕÇÿ¸¥\#ö`À
2ÕÒÕIF›e
```

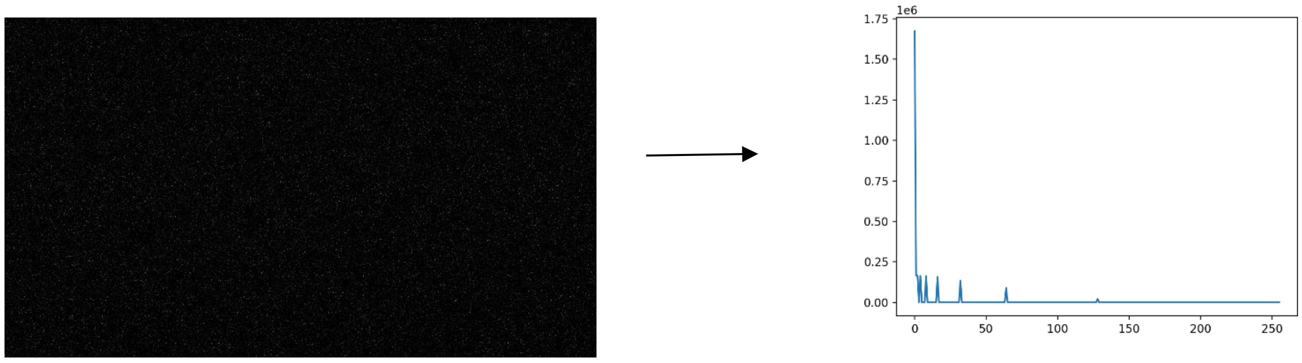
Comparison and Evaluation of result

To assess the security, we used the histogram to compare our encrypted image with an unencrypted image we found online and with our original image.

a. Comparison of our encrypted image with our original image:

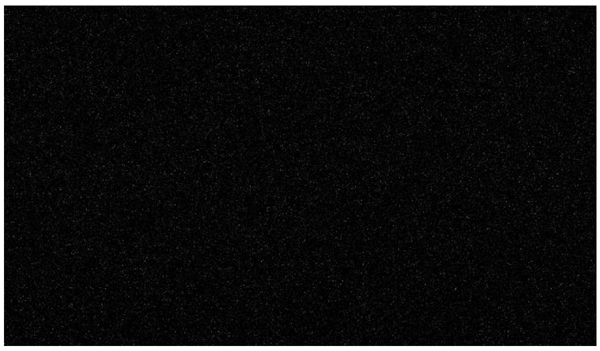
Disney · PIXAR
WALL·E
THE VIDEO GAME

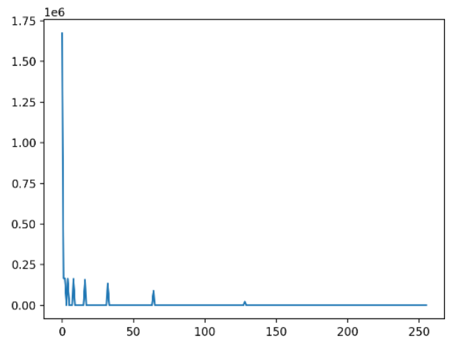


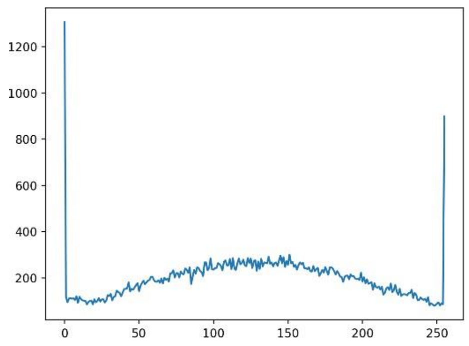


We can see the histograms to the left of both the images, and they are nothing in common, so the deviation of the encrypted image is a lot from the original one.

b. Comparison of our encrypted image with an encrypted image picked from net:







This comparison is made to demonstrate why our method offers greater security than the current system. There are various studies that explore the relationship between an encrypted picture and a histogram. As we can see, they recommend that an encrypted image's histogram should be consistent and horizontally flat.

Conclusion

We investigated a number of encryption techniques and came to the conclusion that AES offers the highest level of security. In the AES method, a user may choose between a 128, 192, or 256 bit planar text, making it user-friendly as well. The complexity increases with the number of rounds, which increases security. It has been shown to be more dependable than DES. AES is allegedly more secure than RSA, according to several academic articles. RSA is substantially slower and requires more work than AES.
Only modest quantities of data are typically encrypted using this method.

- Image encryption using modified aes with bit permutation thus increasing speed of encryption

- To improve security input image is passed through bit slicing and transposition of sliced images algorithms and then taken as input for image encryption using modified aes

Future Goals

Because of the slicing, the final output image turns out to be grayscale. Our long-term objective is to gather and restore the RGB values.

References

[1] K. R. Saraf, V. P. Jagtap, and A. K. M ishra, “Text and Image Encryption Decryption Usinş Advanced Encryption Standard,” Int. J. Emerg. Trends Technol. Comput. Sci., vol. 3, no. 3, pp. 118– 126, 2014.

[2] A. Makhmali and H. M . Jani, “Comparative Study On Encryption Algorithms And Proposing A Data Management Structure,” Int. J. Sci. Technol. Res., vol. 2, no. 6, pp. 42–48, 2013.

[3] H. O. Alanazi, B. B. Zaidan, A. A. Zaidan, H. A. Jalab, M. Shabbir, and Y. Al-Nabhani, “New Comparative Study Between DES, 3DES and AES within Nine Factors,” J. Comput., vol. 2, no. 3, pp. 2151–9617, 2010.

[4] S. El Adib and N. Raissouni, “AES Encryption Algorithm Hardware Implementation: Throughput and Area Comparison of 128, 192 and 256-bits Key,” Int. J. Reconfigurable Embed. Syst., vol. 1, no. 2, pp. 67–74, 2012.

[5] D. F. Garcia, “Performance Evaluation of Advanced Encryption Standard Algorithm,” in 2015 Second International Conference on Mathematics and Computers in Sciences and in Industry (MCSI), 2015, pp. 247–252.

[6] P. Patil, P. Narayankar, Narayan D.G., and Meena S.M., “ A Comprehensive Evaluation oi Cryptographic Algorithms: DES, 3DES, AES, RSA and Blowfish,” Procedia Comput. Sci., vol. 78, no December 2015, pp. 617– 624, 2016.

[7] P. Kumar and S. B. Rana, “Development of modified AES algorithm for data security ,” Optik (Stuttg)., vol. 127, no. 4, pp. 2341–2345, 2016.

[8] R. Rejani and D.V Krishnan, “Study of Symmetric key Cryptography Algorithms,” Int. J. Comput Tech., vol. 2, no. 2, pp. 45–50, 2015.

[9] P. V Kinge, S. J. Honale, and C. M. Bobade, “Design of AES Pipelined Architecture foı Image Encryption / Decryption Module,” Int. J. Reconfigurable Embed. Syst., vol. 3, no. 3, pp. 114– 118, 2014.

[10] M. R. Doomun, K. M . S. Soyjaudah, and D. Bundhoo, “Energy consumption anc computational analysis of Rijndael-AES,” 2007 3rd IEEE/IFIP Int. Conf. Cent. Asia Internet, ICI 2007, 2007.

[11] K. Rahimunnisa, M . Priya Z ach, S. Suresh Kumar, and J. Jayakumar, “Efficient techniques foı the implementation of AES SubByte and MixColumn transformations,” in Advances in Intelligent Systems and Computing, 2012, vol. 176 AISC, no. VOL. 1, pp. 497–506.

[12] S. Rehman, S. Q. Hussain, W.Gul, and Israr, “Characterization of Advanced Encryption Standard (AES) for Textual and Image data,” Int. J. Eng. Comput. Sci., vol. 5, no. October, pp. 18346–18349, 2016.

[13] N. Tyagi and Priyanka, “A Survey on Ensemble of Modifications on AES Algorithm,” J. Basic Appl. Eng. Res., vol. 1, no. 7, pp. 19–24, 2014.

[14] V. C. Koradia, “Modification in Advanced Encryption,” J. Information, Knowl. Res. Comput Eng., vol. 2, no. 2, pp. 356–358, 2013.

[15] F. V Wenceslao, B. D. Gerardo, and B. I. T . Tanguilig, “ Modified AES Algorithm Using Multiple S-Boxes,” in Second International Conference on Electrical, Electronics, Computer Engineering and their Applications (EECEA2015), 2015, vol. 5, no. 1, pp. 1–9

[16] S. Anwarul and S. Agarwal, “Image enciphering using modified AES with secure key transmission,” Commun. Comput. Syst. - Prasad al., pp. 137–142, 2016.

[17] M. A. El-wahed, S. M esbah, and A. Shoukry, “Efficiency and Security of Some Image Encrypt ion Algorithms,” Proc. World Congr. Eng., vol. I, pp. 4–7, 2008.

[18] Y.-Q. Zhang and X.-Y. Wang, “Analysis and improvement of a chaos-based symmetric image encryption scheme using a bit-level p ermutation,” Nonlinear Dyn., vol. 77, no. 3, pp. 687–698, Aug. 2014.

[19] E. Thambiraja, G. Ramesh, and R. Umarani, “A Survey on Various Most Common Encrypt ion Techniques,” Int. J. Adv. Res. Comput. Sci. Softw. Eng., vol. 2, no. 7, pp. 226–233, 2012.

[20] H. Ali-Pacha, N. Hadj-Said, A. Ali-Pacha, M . Mamat, and M . A. Mohamed, “An Efficient Schema of a Special Permutation Inside of Each Pixel of an Image for its Encryption,” Indones. J. Electr. Eng. Comput. Sci., vol. 11, no. 2, 2018.