



VIT[®]
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

❖ SCHOOL OF INFORMATION TECHNOLOGY AND ENGINEERING

• Winter Semester 2022-23

Review Report

**IMAGE WATERMARKING AND ENCRYPTION USING IMAGE
PROCESSING TECHNIQUES**

- Information Security Management – **CSE3502**
- Class No: VL2022230502002
- Slot: F1+TF1

❖ **Faculty:** Prof Sumaiya Theseen I

Submitted By:-

SNO	Reg No:	NAME
01	20BIT0095	R.SAI PRANEETH
02	20BIT0239	C DHANUSH
03	20BIT0258	VANAM JASWANTH
04	20BIT0243	PAVAN KUAMR
05	20BIT0223	Mooli Gokuleswar Reddy

Abstract:

Data production for transmission using any form of multimedia, including digital images is becoming more and more intriguing. Numerous techniques, including cryptography, are employed to maintain the secrecy, integrity, and confidentiality of sensitive information as well as to prevent unauthorized access. By transforming the original data into cipher data, cryptography protects it from being decrypted or altered when it is received by the recipient. To achieve the above mentioned ciphering we used two type of methods as hybrid algorithm one is the watermarking of the image and the other one is the encryption of the image this is mentioned clearly in the below paragraph.

Rubik's Cube Algorithm is used for the encryption of image after applying the water marking. For Watermarking we will use separate Techniques. We will use hybrid techniques like after applying the watermark for the image we will encrypt the image so that the privacy of the content in the image is protected. The main thing we want to achieve by doing this project is privacy and the integrity will be maintained for the image being transmitted in the public networks.

Data are created and transmitted across networks for multimedia applications. These multimedia files contain information and should not be accessed by unauthorized individuals. Therefore, image security and privacy are now key communication issues in the contemporary context. Using a modified Rubik's cube technique, we proposed an advanced encryption scheme in this paper. Two secret keys that are produced using the logistic function and the shift register approach, respectively, are used to first scramble the original image. The scrambled image is then once more combined utilizing a variety of methods using the XOR operator.

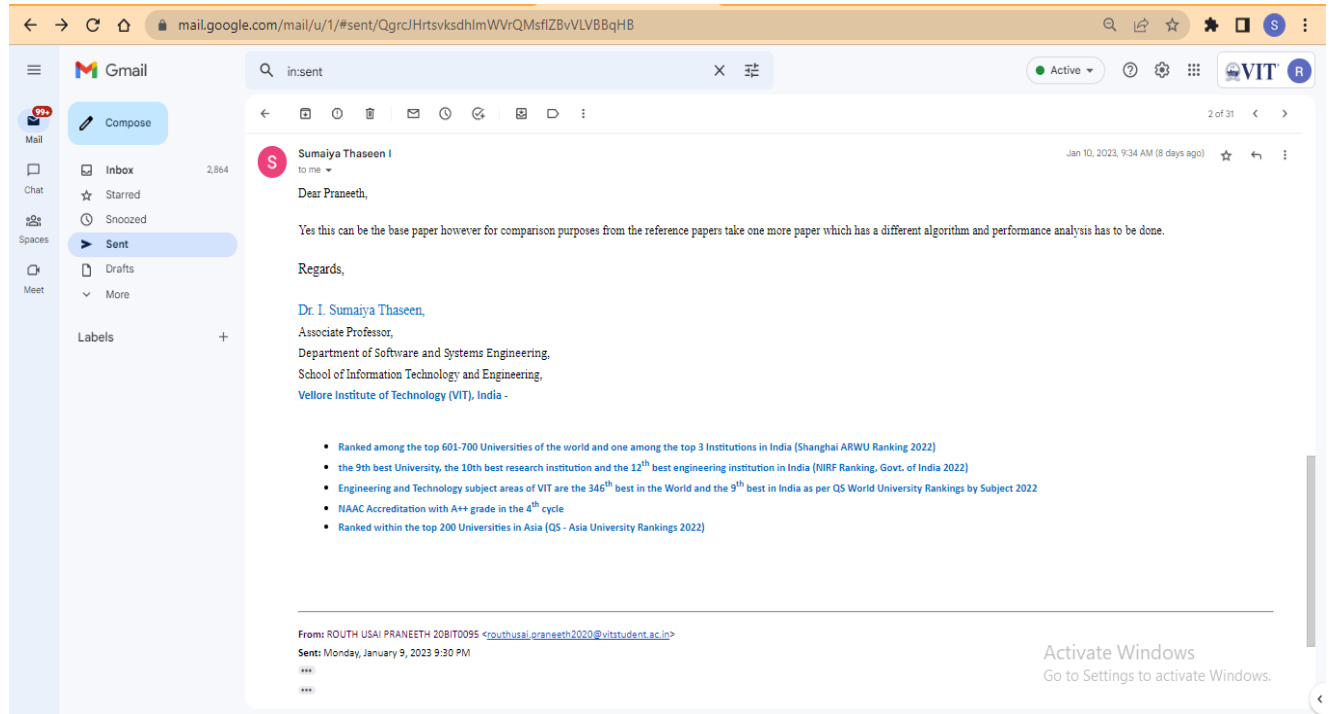
This is the Drive Link for the Base Paper which we have chosen for the encryption of the Image.

[Link](#)

➤ Keywords:

Rubik's Cube Algorithm, Cryptography, Encryption, Watermarking, Keys for Encryption

- Snapshot of the Mail regarding the approval of the Base paper mentioned above.



Literature Survey:

[1] A Secure Image Encryption Algorithm Based on Rubik's Cube Principle:

The Rubik's cube principle is the foundation of the algorithm given in this research, which permutes image pixels. An exclusive kind of XOR technique is used to make it difficult to distinguish between the original and encrypted images. In-depth analysis of experimental tests shows the robustness of the system and various advantages against attackers. Performance evaluation tests also show how highly secure the suggested picture encryption technique is. Additionally, it has quick encryption and decryption capabilities, making it appropriate for Internet-based real-time encryption and transmission applications.

Algorithm	Advantages	Issues/Drawbacks
Rubik's Cube Image Encryption Rubik's Cube Decryption Algorithm	<p>This encryption algorithm is very fast compared to other algorithms</p> <p>This encryption algorithm is highly robust against entropy attacks. The encrypted images are hardly correlated. There is no visual similarity between original ones and their encrypted images.</p>	<p>This image encryption algorithm resists cropping attacks lightly. Random noise attacks are seriously harmful for the decrypted images</p>

[2] A novel colour image encryption algorithm based on chaos –

This research proposes a revolutionary chaos-based colour image encryption technique. The R, G, and B components of a colour image were simultaneously encrypted by the authors using a chaotic system, which caused the three components to interact. In this way, the algorithm's security can be improved while the correlations between the R, G, and B components are lowered. Simulation findings demonstrate that the suggested approach can successfully encrypt colour images and withstand a variety of common attacks.

Algorithm	Advantages	Issues/Drawbacks
Chaos based image encryption algorithm.	The proposed algorithm is resistant to various typical attacks.	The drawbacks for this paper is not secure.

[3] Rubik's cube principle-based image encryption algorithm implementation on mobile devices:

In this research, a communication system between numerous mobile devices with imaging sensors—which encrypt the images using the Rubik cube encryption algorithm—and a server is

implemented. Investigated are the algorithm's effectiveness and applicability for mobile devices. The article ran a number of tests to evaluate how well the algorithm worked on a mobile device. The first test concerned the devices' processing (and encryption) speed.

The processing speed of the server is contrasted with this. Performance for a single process/single thread is compared. The server can process more photos simultaneously than a mobile device since it has many times as many CPUs.

Algorithm	Advantages	Issues/Drawbacks
Rubik's cube algorithm for image encryption	The processing time is low and on lower end mobile devices and allows low powered devices to perform this encryption method efficiently. Tests also showed that the algorithm speed is comparable with the AES128 encryption algorithm	Large size images can decrease the performance.

[4] Chaotic Image Encryption Algorithm Based on Fractional Order Scrambling Wavelet Transform and 3D Cyclic Displacement Operation:

The chaotic picture encryption method put forth in this research uses a fractional wavelet transform. First, the image is pre-processed as a signal using the fractional wavelet transform. High-frequency components are produced with low frequency components following three-level fractional wavelet decomposition. Second, the chaotic sequence is obtained by entering the initial value into the Lorenz system and then using the hash function to generate the chaotic system's initial value for the plaintext operation. The chaotic sequence is then scrambled for the high frequency components and 3D cyclic displacement scrambling is used for the low frequency components.

Algorithm	Advantages	Issues/Drawbacks
Chaotic Encryption Algorithm with 3D Cyclic Displacement Operation.	Key space is larger than 2^{100} . Which is enough to withstand brute force attacks? If a cryptographic system is highly sensitive to keys, it is a good key system. The algorithm in this paper is extremely sensitive to the key.	The ciphertext images obtained in this paper are less likely to leak information and have better ability to resist statistical analysis attacks. But still not yet very secure.

[5]A chaos-based image encryption algorithm using Rubik's cube and prime factorization process –

In order to safely transmit multimedia information (pictures) across an untrusted channel, such as adaptive image content (i.e., plain image related) based initialization, an unique Rubik's cube-based pixel level scrambling method and simple XOR based diffusion are proposed in this study.

To achieve high plain image sensitivity and fend off threats connected to plain images, random value generation is introduced. The beginning vectors of the Henon map are derived from this random value, and this procedure is repeated to derive the key sequences to be applied across the Rubik's cube row and column confusion processes. The key generation technique based on prime factorization to be used in diffusion also uses the same random seed.

Algorithm	Advantages	Issues/Drawbacks
Image encryption algorithm using Rubik's cube and prime factorization process	This method is shown to be secure against differential attacks. Also, from the simulations, it is shown that the proposed methodology has good key space, high key sensitivity, and uniform distribution of cipher image pixels	Large size images can decrease the performance.

[6] 3-D Image Encryption Based on Rubik's Cube and RC6 Algorithm –

To create 3D encryption of a collection of photos, a unique encryption technique based on the 3-D Rubik's cube is proposed in this study. For independently encrypting several photos, this proposed encryption technique starts with RC6. The 3-D Rubik's cube is then used to further encrypt the images that were previously encrypted. The Rubik's cube's faces are made from RC6-encrypted images. The Rubik's cube algorithm adds a degree of permutation, while the RC6 technique adds a degree of diffusion to the ideas of picture encryption. The simulation results show that the suggested encryption algorithm works well and has high levels of security and reliability.

Algorithm	Advantages	Issues/Drawbacks
Rubik's Cube and RC6 Algorithm	Results show that the proposed encryption algorithm is efficient, and it exhibits strong robustness and security.	Low resolution images may decrease the performance of the method.

[7] Diffusion–substitution based grey image encryption scheme-

This study proposes an encryption technique for grayscale photographs using a 128-bit secret key. Initially, the mixing process reduces the visual quality of the image. The resulting image is divided into dynamic blocks that depend on keys, and these blocks are then put through key-dependent diffusion and substitution procedures. The encryption algorithm makes use of a total of sixteen cycles. The suggested method has a high encryption rate and is easy to use. Results from simulation experiments have been provided to attest to the effectiveness and high security aspects of the suggested solution.

Algorithm	Advantages	Issues/Drawbacks
Encryption algorithm for grey images using a secret key of 128-bits size	The proposed algorithm is show cryptanalytic, statistical and brute-force attacksn to be resistant against	Key must be chosen carefully as key used in the image cipher cannot be too long nor too short. A larger key decreases the encryption rate and is not preferred for real time image transmission, and a shorter key result in an easy cryptanalysis.

[8]Secure Communication Based on Rubik's Cube Algorithm and Chaotic Baker Map –

Digital and multimedia applications in the fields of art, entertainment, advertising, business, education, and training generate more multimedia data that is shared across networks and may contain sensitive information that should not be accessed by unauthorised parties. Maintaining the validity, confidentiality, integrity, security, and privacy of images has thus grown to be a top priority for image storage and communication. This paper proposes an effective Rubik's cube-based and chaotic Baker map-based image encryption method. There are two layers to it. Preprocessing is done on the first layer, which helps the image's quality. The Rubik's cube principle is applied in the second layer.

Algorithm	Advantages	Issues/Drawbacks
Chaotic Baker Map Rubik's Cube Algorithm	It is simple and achieves good permutation and diffusion mechanisms The algorithm provides user flexibility by providing Encryption to a wide range of images. The provides the flexibility of choosing standard format images of any shape and size.	software It uses more time for encryption. But time can be manageable by using hardware with high processing speed.

[9]A light weight secure image encryption scheme based on chaos & DNA computing –

For secure image communication, a new, lightweight cryptographic technique was put forth in this study. In the suggested method, the plain image is first permuted using a series of pseudo random numbers (PRN), and then encrypted using calculations involving deoxyribonucleic acid (DNA). A pseudo-random number generator (PRNG) built on a cross linked chaotic logistic map generates two PRN sequences utilising two sets of keys. While the second PRN sequence is utilised to create random DNA sequences, the first PRN sequence is used to permute the simple image. To boost security, the number of permutation and encryption rounds can be adjusted.

Algorithm	Advantages	Issues/Drawbacks
light weight secure cryptographic scheme.	The proposed methodology is resistant against statistical attacks. Permutation is also done on a plain image which gives better performance and quality	The methodology can be further improved to work on applications related to Iot.

[10] An Efficient Image Encryption Scheme Based on the LSS Chaotic Map and Single S-Box-

An effective and safe chaotic S-Box based picture encryption technique is presented in this research.

First, using a chosen-plaintext attack, the authors successfully broke the cryptosystem by cryptanalyzing a multiple chaotic S-Boxes-based picture encryption technique (CPA). The research also introduces a brand-new picture encryption method based on a singular S-Box and a novel compound chaotic map. A unique discrete compound chaotic system called the Logistic-Sine system (LSS), which has a wider chaotic range and superior chaotic features, is proposed in the new scheme. The outcomes of the simulation and the security analysis attested to the efficiency of the suggested image encryption method.

Algorithm	Advantages	Issues/Drawbacks
LSS Chaotic Map and Single S-box	The simulation results and security analysis verified the effectiveness of the proposed image encryption scheme. The proposed algorithm has efficiency advantages, showing that it has better application potential in real-time image encryption.	NA

[11] Multistage Image Encryption using Rubik's Cube for Secured Image Transmission-

Together with the Rubik's cube encryption technique, the LSB Steganography algorithm offers us exceptional security and safeguards the image during transmission. Image resolution essentially stays the same. When a message is included in the image, it barely changes. The magic cube rotation technique also protects the image. Additionally, the pace of data embedding into images is very great.

The process of steganography involves concealing data behind an image. A text or image file in one format is concealed in another text or image file in the same format or in a different format. Nowadays, the aim of protected transmission fails since hackers may easily hack the data transmitted. Traditional data transmission methods include encryption, scrambling, watermarking.

Steganography, for example; encryption entails converting data from one format to another before sending. When the intruders are aware of the decryption algorithm, the data is readily available to them. The majority of encryption methods are predictable. a fresh approach to image.

Algorithm	Advantages	Issues/Drawbacks
Multi stage image Encryption using Rubik's Cube Algorithm	Provides high security. Protects the image while Transmission.	Requires high computational performance.

[12] A Novel Color Image Encryption Algorithm Based on Three-Dimensional Chaotic Maps and Reconstruction Techniques –

This paper suggests a novel color image encryption strategy that makes use of three-dimensional chaotic maps and a few data reconstruction approaches. The diffusion step of the encryption technique alters the pixel value of the plain image using a three-dimensional chaotic logistic map. To deal with the location of the image pixels during the confusion phase, a three-dimensional chaotic Cat map is used. Additionally, a number of data reconstruction techniques are employed to enhance the results of the encryption. Also provided is the matching decryption algorithm. Our proposed method breaks the correlation between picture pixel layers, offers a large key space, and successfully fends off differential and statistical attacks. The effectiveness and simplicity of our approach are important advantages.

Algorithm	Advantages	Issues/Drawbacks
The algorithms include Encryption and decryption methods of color image in detail. The whole encryption algorithm involves the diffusion process and the confusion process.	In the input colour image, nearby pixels have a strong correlation, whereas in the output image, neighbour pixels have a much weaker correlation. High property of resist statistical attack. Good local information entropy test results.	This is greatly sensitive to the original input image to cope with the differential attacks. Keys depend on the size of the original input image.

[13] A Novel Double Image Encryption Algorithm Based on Rossler Hyperchaotic System and Compressive Sensing –

This work proposes a novel double-image compression encryption method that combines Rossler hyper chaos and compressive sensing. The proposed method uses index confusion and compressive sensing to scramble and compress the sparse matrices obtained by performing a two dimensional discrete wavelet transform on two plain images. The final noise-like cypher image is produced by hiding half of the encrypted image in the alpha channel of the other half of the

encrypted image. One cypher picture corresponds to one key is a concept that is realised through the creation of a special key generation procedure. Finally, simulation outcomes demonstrate the proposed double-image encryption system's high transmission efficiency.

Algorithm	Advantages	Issues/Drawbacks
Rossler Hyper chaotic system	The compression rate is as high as 400%. The decrypted images have a good visual effect	The time complexity of generating chaotic sequences and measuring matrix is $\theta(N^2)$ and $\theta(CR \times N^2)$, respectively which is quite large

[14] A Plaintext-Related Dynamical Image Encryption Algorithm Based on Permutation Combination-Diffusion Architecture –

This work proposes a brand-new, highly secure dynamical picture encryption technique. In contrast to the traditional permutation-diffusion encryption architecture, we added the combination operation before the diffusion, which uses the 3D-ILM to create a dynamic interference matrix the same size as the grayscale plaintext image. The permuted image is then combined with the interference matrix.

Algorithm	Advantages	Issues/Drawbacks
dynamical image encryption algorithm based on permutation-combination diffusion architecture	(1) The proposed algorithm is highly related to the plaintext image. (2) By adding the combination process, the diffusion and permutation process are correlated, which can resist the separate attacking on diffusion and permutation process. (3) The cipher image is dynamic changing even the plaintext image is unchanged.	the proposed encryption algorithm has extremely high sensitivity to the secret keys.

[15] **A robust image encryption algorithm based on Chua's circuit and compressive sensing –**

First, using a Haar wavelet, the plaintext picture is divided into approximate and detail components. When Chua's circuit and the Logistic map are combined to create a local binary patterns (LBP) operator-based chaotic sequence, the approximation component is then diffused. The chaos-combined asymptotic deterministic random measurement matrices (CADRMM), which are used to measure the detail components in various compression ratios, are then created by applying the Lissajous map.

Algorithm	Advantages	Issues/Drawbacks
No specific algorithm the measurement matrix CADRMM is mainly generated by Lissajous map and the logistic map SHA-256 is applied to produce the keys.	the test results prove that the measurement matrix of this work owns good performance and effectiveness in compressive sensing. Simultaneously, it has no negative effect on image reconstruction.	The correlation between pixels of plain-text is very strong but that of the corresponding encrypted image is really weak.

[16] **A survey of digital image watermarking techniques**

There has been a lot of interest in studies about watermarking, which is a part of the information hiding topic. Numerous projects are now being worked on in various sections of this discipline. While watermarking is used for content protection, copyright management, content authentication, and tamper detection, steganography is utilised for covert communication. In this article, we provide a thorough overview of both recently proposed and existing steganographic and watermarking techniques. We categorise the methods according to the various domains in which data is incorporated. We only include photographs in the survey.

Algorithm	Advantages	Issues/Drawbacks
Robust watermarking algorithm within the DCT, DWT and DF domain.	DFT and DCI are full frame transform, and hence any change in the transform coefficients affects the entire image except if DCT is implemented using a block based approach. However, DWT has spatial frequency locality, which means if signal is embedded it will affect the image locally. Hence a wavelet transform provides both frequency and spatial description for an image.	Computational complexity is more. It only takes 54 multiplications to compute DCT for a block of 8x8, unlike wavelet calculation depends upon the length of the filter used, which is at least 1 multiplication per coefficient.

[17] A Review of Image Watermarking Applications in Healthcare

we emphasise the complementing function of watermarking in terms of medical information management and security (integrity, authenticity, etc.). We look at some sample applications of watermarking. We draw the conclusion that watermarking has carved out a specific place in healthcare systems as a tool for medical data security, safe handling, and dissemination. Medical professionals continue to place a high priority on record preservation and diagnostic fidelity.

Algorithm	Advantages	Issues/Drawbacks
This paper is the review of Image Watermarking Applications in Healthcare	advantage of reversible watermark is that the image becomes authenticated.	It becomes difficult to change the image format

[18] Image watermarking using soft computing techniques:

Image watermarking techniques are used to provide copyright protection and verify ownership of media/entities. This technique refers to the concept of embedding of secret data/information of an owner in a given media/entity for determining any ownership conflicts that can arise. Many watermarking approaches have been offered by various authors in the last few years. However, there are not enough studies and comparisons of watermarking techniques in soft computing environments. Nowadays, soft computing techniques are used to improve the performance of watermarking algorithms. This paper surveys soft computing-based image watermarking for several applications. We first elaborate on novel applications, watermark characteristics and different kinds of watermarking systems. Then, soft computing based watermarking approaches providing robustness, imperceptibility and good embedding capacity are compared systematically.

Algorithm	Advantages	Issues/Drawbacks
Machine learning. CNN . SVM . Neural networks. Genetic algorithm. Fuzzy logic. PCA	secret data is hidden within the host and exchanged without generating any kind of visible alert to the attackers.	The SVM classifier is used to determine the ROI-RONI part of host image to achieve the better robustness of watermark. However, it required more computational time during training phase of SVM classifier

[19] Robust image watermarking in the spatial domain

The necessity for the protection of intellectual property rights on photographs has become urgent due to the quick development of digital image processing and transmission systems. This work presents a copyright protection technique based on the obfuscation of a digital watermark, a "invisible" signal, in the image. By slightly adjusting the intensity of randomly chosen image pixels, watermark casting is carried out in the spatial domain. The process of detecting a

watermark compares the mean intensity value of the marked pixels to that of the unmarked pixels without requiring the presence of the original image. For this, statistical hypothesis testing is employed.

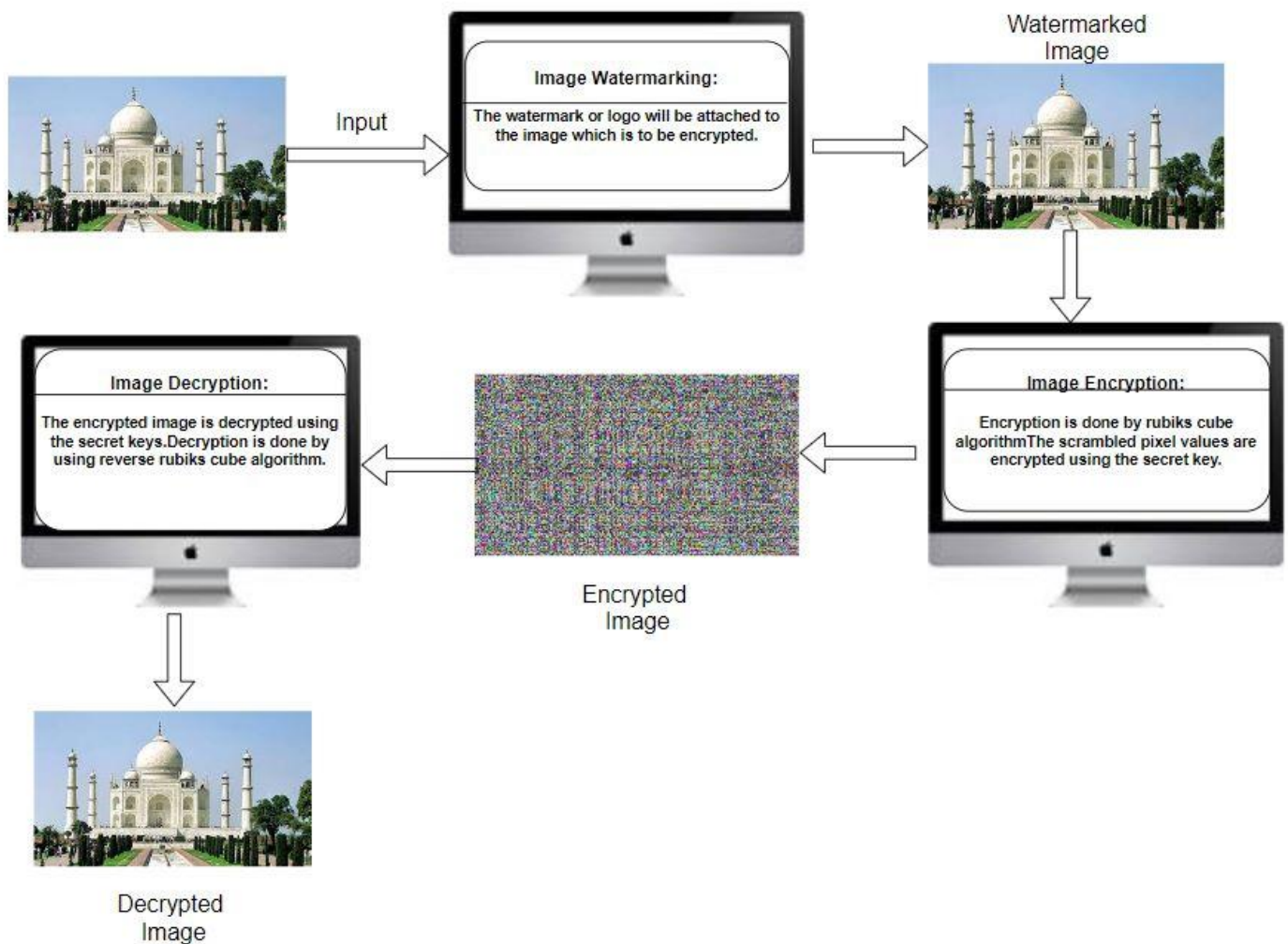
[20] Fair benchmark for image watermarking systems

Since the early 90s a number of papers on 'robust' digital watermarking systems have been presented but none of them uses the same robustness criteria. This is not practical at all for comparison and slows down progress in this area. To address this issue, we present an evaluation procedure of image watermarking systems. First we identify all necessary parameters for proper benchmarking and investigate how to quantitatively describe the image degradation introduced by the watermarking process. For this, we show the weaknesses of usual image quality measures in the context watermarking and propose a novel measure adapted to the human visual system. Then we show how to efficiently evaluate the watermark performance in such a way that fair comparisons between different methods are possible. The usefulness of three graphs: 'attack vs. visual-quality,' 'bit-error vs. visual quality,' and 'bit-error vs. attack' are investigated. In addition the receiver operating characteristic (ROC) graphs are reviewed and proposed to describe statistical detection behavior of watermarking methods. Finally we review a number of attacks that any system should survive to be really useful and propose a benchmark and a set of different suitable images.

- **High Level Design:**

The high-level design of Image encryption using the modified Rubik's cube principle involves the following steps:

1. Image Watermarking: The watermark or logo will be attached to the image which is to be encrypted.
2. Image preprocessing: The input image is converted into a square matrix of fixed size.
3. Rubik's cube transformation: The square matrix is transformed using the Rubik's cube principle to scramble the pixel values.
4. Key generation: Two secret keys are generated, which is used to encrypt and decrypt the image.
5. Encryption: The scrambled pixel values are encrypted using the secret key.
6. Decryption: The encrypted image is decrypted using the secret keys.
7. Rubik's cube inverse transformation: The decrypted pixel values are transformed back to their original positions using the Rubik's cube inverse transformation.
8. Output: The final output is the decrypted image.



- **Low Level Design:**

The low-level design of Image encryption using the modified Rubik's cube principle involves the following steps:

1. **Image Watermarking:** The watermark or logo will be attached to the image which is to be encrypted. The logo or watermark will be of fixed size and based on the input image size logo will be adjusted.
2. **Image preprocessing:**
 - a. Convert the input image into a square matrix of fixed size.
 - b. Convert the pixel values of the image into array form.
3. **Key generation:**

- a. Two keys are generated namely K_r and K_c having length m and n , respectively. Key K_r is generated using functions in python and generates random integers.

4. Modified Rubik's cube transformation:

- Then, in the image perform image shifting row-wise for each row as follows:
 - a. Find the sum of all the elements of the row and take it's modulo 2 denoted by $M(i)$.
 - b. Then based on the positions of $K_r(i)$, left or right circular-shift operation is performed on row i , according to the following condition: If $M(i) = 0$, then right circular-shift otherwise left circular-shift.
- Now in the image, perform image shifting column-wise for each column as follows:
 - a. Find the sum of all elements of the column and take it's modulo 2 denoted by $M(j)$.
 - b. Then based on the positions of $K_c(j)$, down or up circular-shift operation is performed on column j , according to the following condition: If $M(j) = 0$, then up circular-shift otherwise down circular-shift.

5. Performing XOR operations

- Now for each row of the image, perform XOR operation of the image row-wise as follows:
 - a. XOR the row with K_c and then circularly right shift K_c by one element.
- Now for each column of the image, perform XOR operation of the image columnwise as follows:
 - a. XOR the column with K_r and then circularly right shift K_r by one element.

6. Encryption:

- a. Apply the modified Rubik's cube transformation to the pixel values in each sub-matrix.
- b. XOR the transformed pixel values with the corresponding key from the secret key.

7. Storing keys in text file which will be automatically written to the file by python.

8. Decryption:

- Initialize $ITER = 0$, increment by 1 after each execution.
- XOR each column with K_r and circularly right shift K_r by one element.
- XOR each row with K_c and circularly right shift K_c by one element.
- Calculate the sum of each column, find modulo 2 as $M(j)$, then up/down circular-shift column j based on the positions of $K_c(j)$ and $M(j)$.

- Calculate the sum of each row, find modulo 2 as M(i), then left/right circular-shift row i based on the positions of Kr(i) and M(i).
- If ITER = ITERMax, encryption process is over, else switch to step 2.

9. Output:

- Convert the binary pixel values back to decimal form.
- Convert the matrix back into an image.

10. Comparison of correlation of images:

- We will find the correlation coefficient of the image before and after the encryption.
- Comparing the correlation coefficients of both the images
- The logic used for the correlation calculation is:

$$r = (co1 - co2) / \text{sqrt}(co3 * co4)$$

This formula is derived from the formula for Pearson's correlation coefficient, where the numerator represents the covariance of the two random variables, and the denominator represents the product of their standard deviations.

11. Comparison of entropy of images:

- We will find the entropy of the image before and after the encryption.
- Comparing the entropy of both the images
- The logic used for the calculation is:

$$e = -\sum(p(g)*\log_2(p(g)))$$

3.2 Information Entropy

Information entropy is a quantity which is used to describe the amount of information which must be coded by a compression algorithm. Entropy can be calculated by the formula given below,

$$H(c) = \sum_{i=0}^{2^L-1} P(c_i) \log_2 \frac{1}{P(c_i)}$$

For 256 bits gray-scale images, theoretical value of entropy should be equal to 8. For this work, entropy of the encrypted image is very close to the ideal value of 8, which indicates the robustness of the proposed algorithm against entropy attacks



DECRYPTED

Image Watermarking

The watermark or logo will be attached to the image

Image Processing

A) Convert the input image into square matrix of fixed size
B) Convert the pixel value of the image into array form

Key Generation

A) Two keys are generated namely kr and kc having length m, n.
B) Key kr is generated using functions in python and generate random integers

Rubick's cube Transformation

A) Then, in the image perform image shifting row-wise
B) Then, in the image perform image shifting column wise for each column

Perform XOR Operations

A) XOR the with kc and then circularly right shift by one element
B) XOR the column with kr and then circularly right shift kr by one element

Output

A) Convert the binary pixel values to decimal form
B) Convert the matrix back into an image

Decryption

1. Just reversing the operations done in the encryption.
2. Getting the keys kr and Kc from the user which are generated in encryption.

Storing Keys in text file which will be writing to the file in py

The keys will be stored in keys.txt file



Encrypted

Encryption

A) Apply the modified Rubik's cube transformation to the pixel values in each sub matrix
B) XOR the transformed pixel values with the corresponding key from the secret key

Comparison of correlation in images

1. This module is
2. used to compare the correlation between the images.
3. Without encryption and with encryption.
$$Y(r,c) = \sum h r' = -h \sum w c' = -w l(r+r', c+c') F(r', c')$$

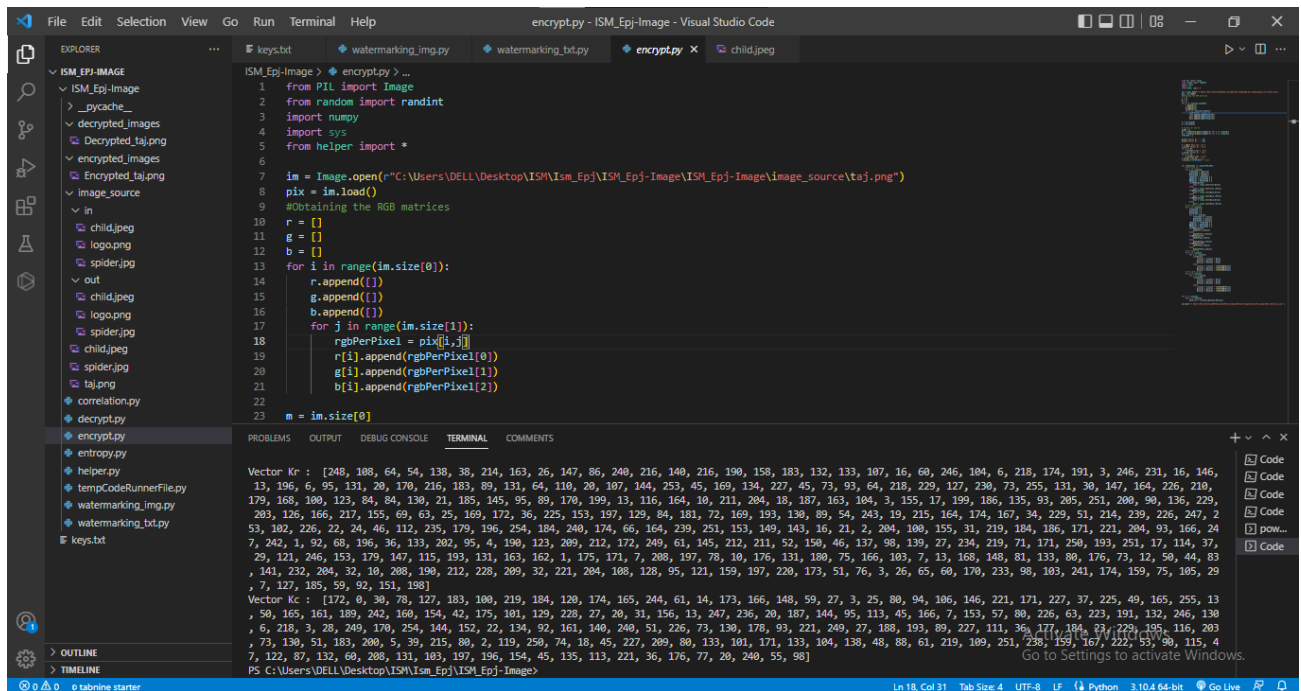
Comparison of Entropy in images

1. This module is
2. used to compare the correlation between the images.
3. Without encryption and with encryption.
$$-\sum(p_i \cdot \log_2(p_i))$$

P-Normalized histogram counts

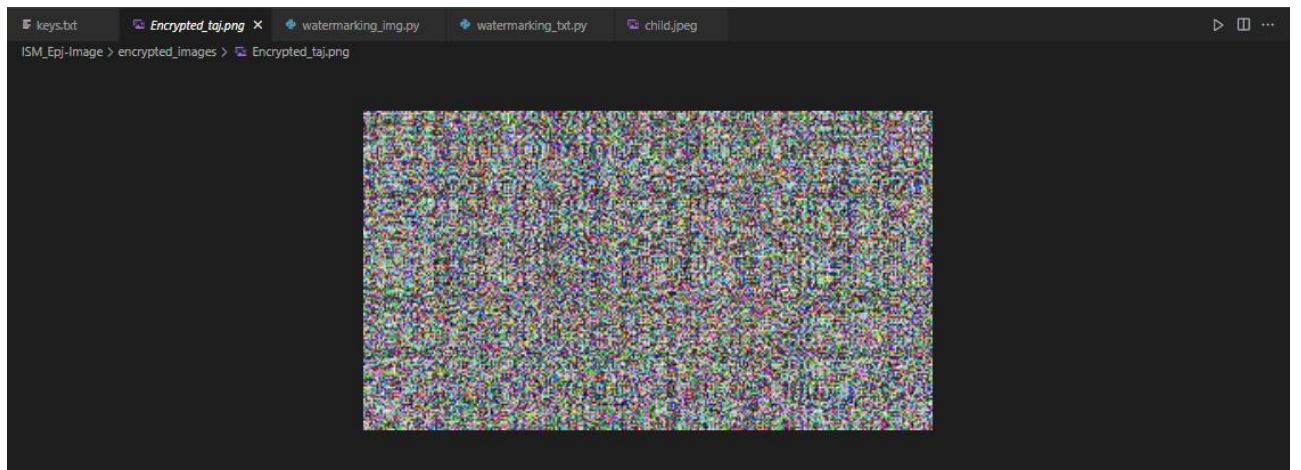
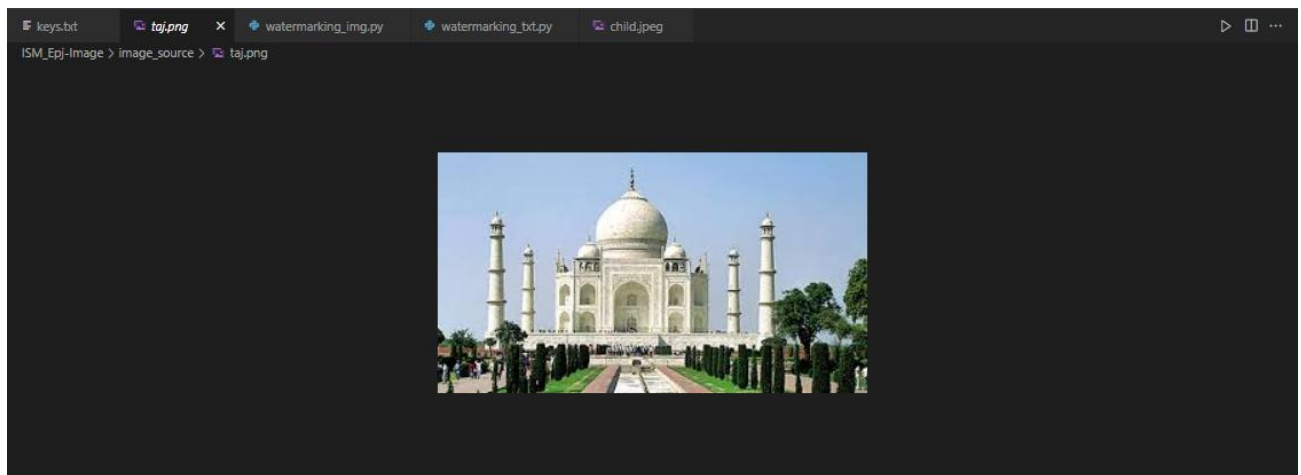
Screenshot of all the modules:

- Encryption of an image:

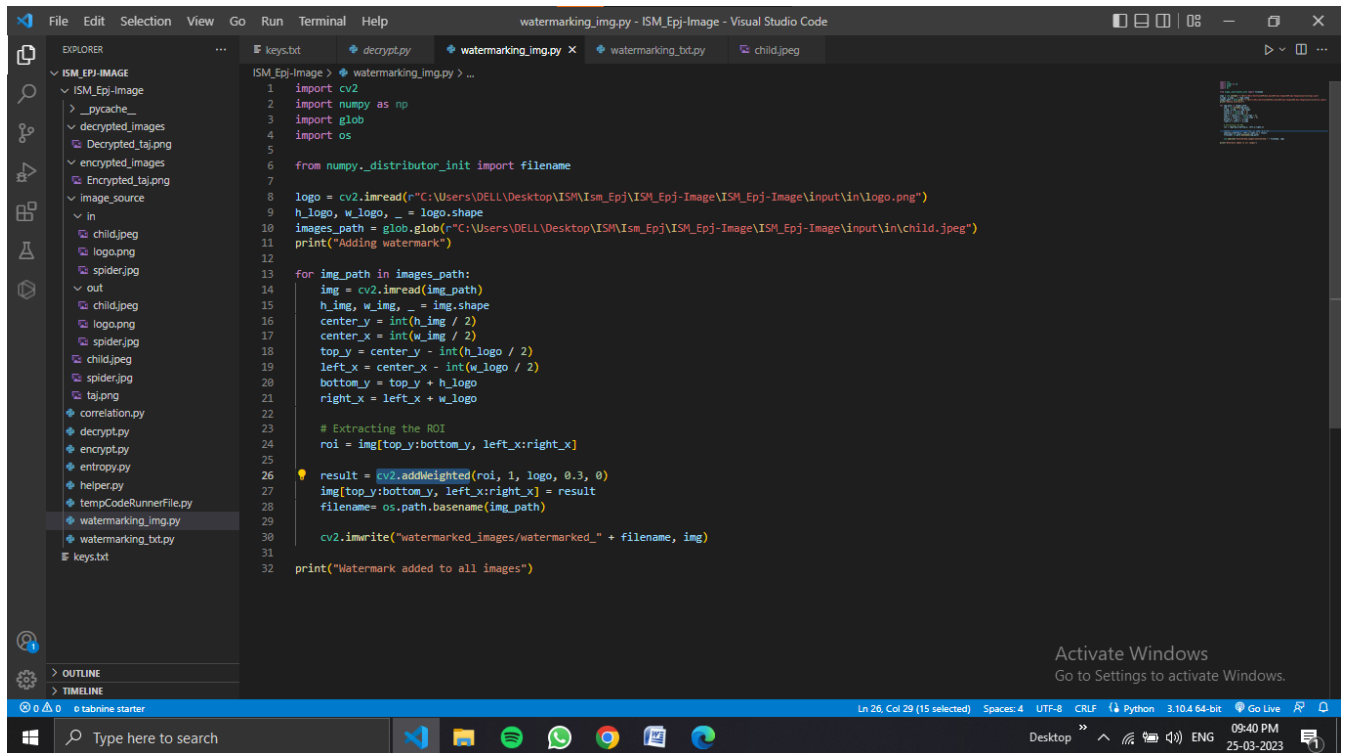


The screenshot shows the Visual Studio Code interface with the following components:

- EXPLORER:** Displays the project structure for 'ISM_Epj-Image'. It includes folders for 'ISM_Epj-Image', '._pycache_', 'decrypted_images', 'Encrypted_taj.png', 'image_source', and 'in'. The 'image_source' folder contains 'taj.png'.
- Code Editor:** Shows the 'encrypt.py' script. The code imports 'Image' from 'PIL', 'randint' from 'random', 'numpy' from 'numpy', and 'sys' from 'sys'. It uses 'helper' for some functions. The script opens 'taj.png' from the 'image_source' folder, loads it, and then iterates over each pixel to perform encryption. The encryption process involves generating random values and applying them to the pixel's RGB channels.
- TERMINAL:** Displays the output of the script. It shows the 'Vector Kr' and 'Vector Kc' values, which are used for the encryption process. The output is a long list of integers representing the encrypted data.



- Watermarking of the Image:

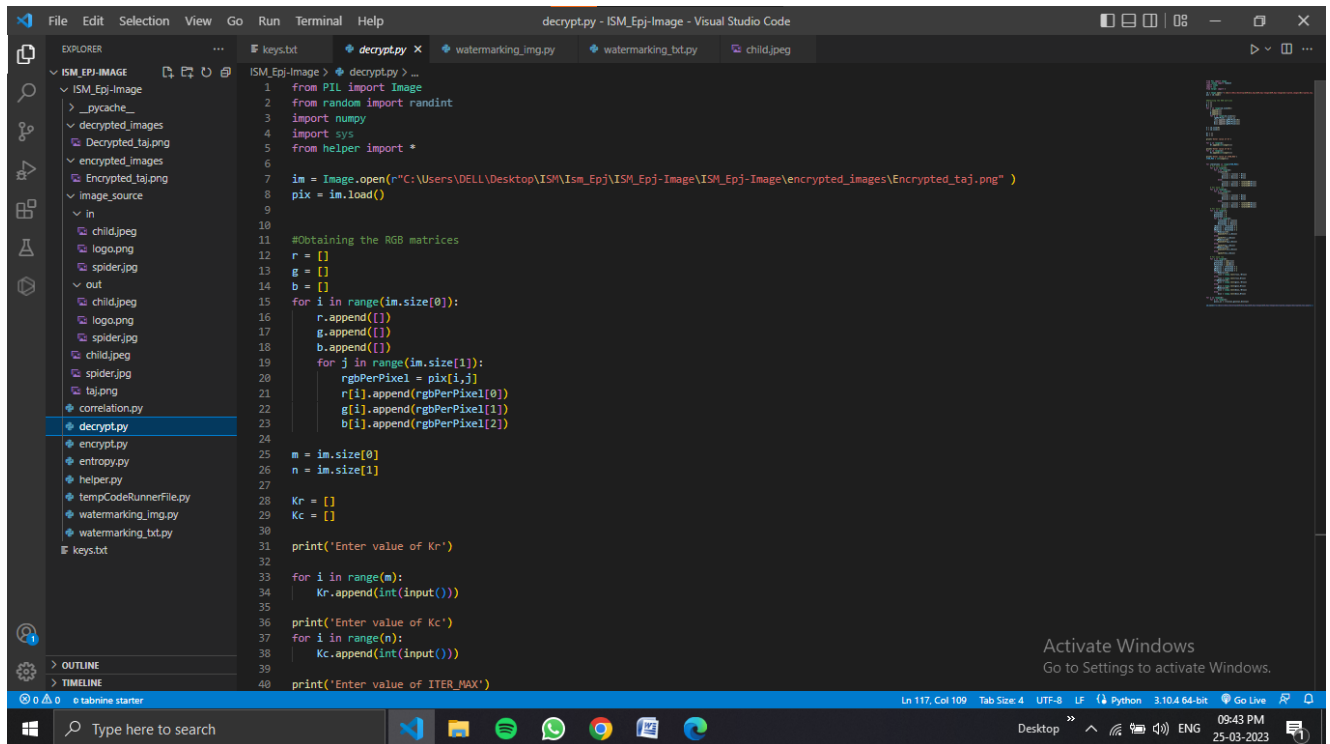


```

1 import cv2
2 import numpy as np
3 import glob
4 import os
5
6 from numpy._distributor_init import filename
7
8 logo = cv2.imread(r"C:\Users\DELL\Desktop\ISM\Ism_Epj\ISM_Epj-Image\ISM_Epj-Image\input\in\logo.png")
9 h_logo, w_logo, _ = logo.shape
10 images_path = glob(r"C:\Users\DELL\Desktop\ISM\Ism_Epj\ISM_Epj-Image\ISM_Epj-Image\input\in\child.jpeg")
11 print("Adding watermark")
12
13 for img_path in images_path:
14     img = cv2.imread(img_path)
15     h_img, w_img, _ = img.shape
16     center_y = int(h_img / 2)
17     center_x = int(w_img / 2)
18     top_y = center_y - int(h_logo / 2)
19     left_x = center_x - int(w_logo / 2)
20     bottom_y = top_y + h_logo
21     right_x = left_x + w_logo
22
23     # Extracting the ROI
24     roi = img[top_y:bottom_y, left_x:right_x]
25
26     result = cv2.addWeighted(roi, 1, logo, 0.3, 0)
27     img[top_y:bottom_y, left_x:right_x] = result
28     filename = os.path.basename(img_path)
29     cv2.imwrite("watermarked_images/watermarked_" + filename, img)
30
31 print("Watermark added to all images")
32

```

- Decryption of an image:

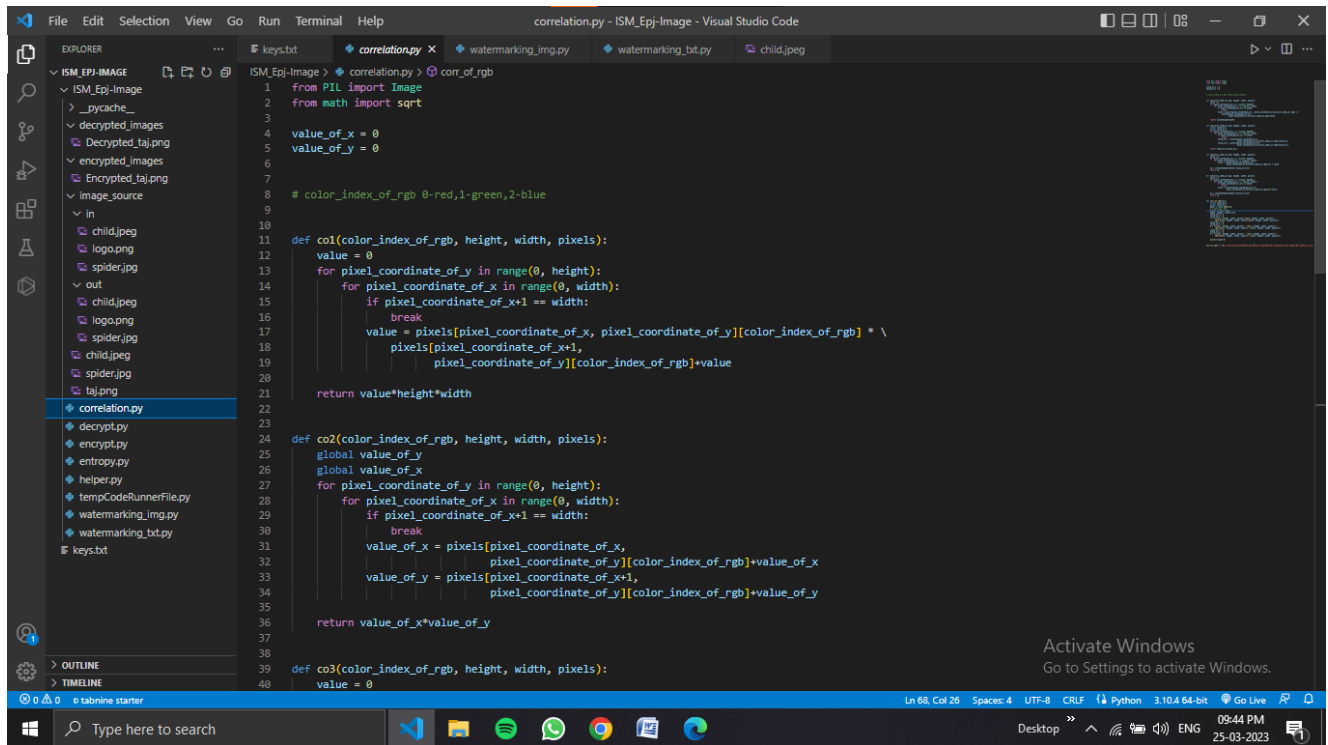


```

1 from PIL import Image
2 from random import randint
3 import numpy
4 import sys
5 from helper import *
6
7 im = Image.open(r"C:\Users\DELL\Desktop\ISM\Ism_Epj\ISM_Epj-Image\ISM_Epj-Image\encrypted_images\Encrypted_taj.png")
8 pix = im.load()
9
10 #Obtaining the RGB matrices
11 r = []
12 g = []
13 b = []
14
15 for i in range(im.size[0]):
16     r.append([])
17     g.append([])
18     b.append([])
19     for j in range(im.size[1]):
20         rgbPerPixel = pix[i,j]
21         r[i].append(rgbPerPixel[0])
22         g[i].append(rgbPerPixel[1])
23         b[i].append(rgbPerPixel[2])
24
25 m = im.size[0]
26 n = im.size[1]
27
28 Kr = []
29 Kc = []
30
31 print('Enter value of Kr')
32
33 for i in range(m):
34     Kr.append(int(input()))
35
36 print('Enter value of Kc')
37 for i in range(n):
38     Kc.append(int(input()))
39
40 print('Enter value of ITER_MAX')

```


- Correlation of an Image:



```

1 from PIL import Image
2 from math import sqrt
3
4 value_of_x = 0
5 value_of_y = 0
6
7 # color_index_of_rgb 0:red,1:green,2:blue
8
9
10
11 def col(color_index_of_rgb, height, width, pixels):
12     value = 0
13     for pixel_coordinate_of_y in range(0, height):
14         for pixel_coordinate_of_x in range(0, width):
15             if pixel_coordinate_of_x+1 == width:
16                 break
17             value = pixels[pixel_coordinate_of_x, pixel_coordinate_of_y][color_index_of_rgb] * \
18                 pixels[pixel_coordinate_of_x+1,
19                     pixel_coordinate_of_y][color_index_of_rgb]+value
20
21     return value*height*width
22
23
24 def co2(color_index_of_rgb, height, width, pixels):
25     global value_of_y
26     global value_of_x
27     for pixel_coordinate_of_y in range(0, height):
28         for pixel_coordinate_of_x in range(0, width):
29             if pixel_coordinate_of_x+1 == width:
30                 break
31             value_of_x = pixels[pixel_coordinate_of_x,
32                             pixel_coordinate_of_y][color_index_of_rgb]+value_of_x
33             value_of_y = pixels[pixel_coordinate_of_x+1,
34                             pixel_coordinate_of_y][color_index_of_rgb]+value_of_y
35
36     return value_of_x*value_of_y
37
38
39 def co3(color_index_of_rgb, height, width, pixels):
40     value = 0

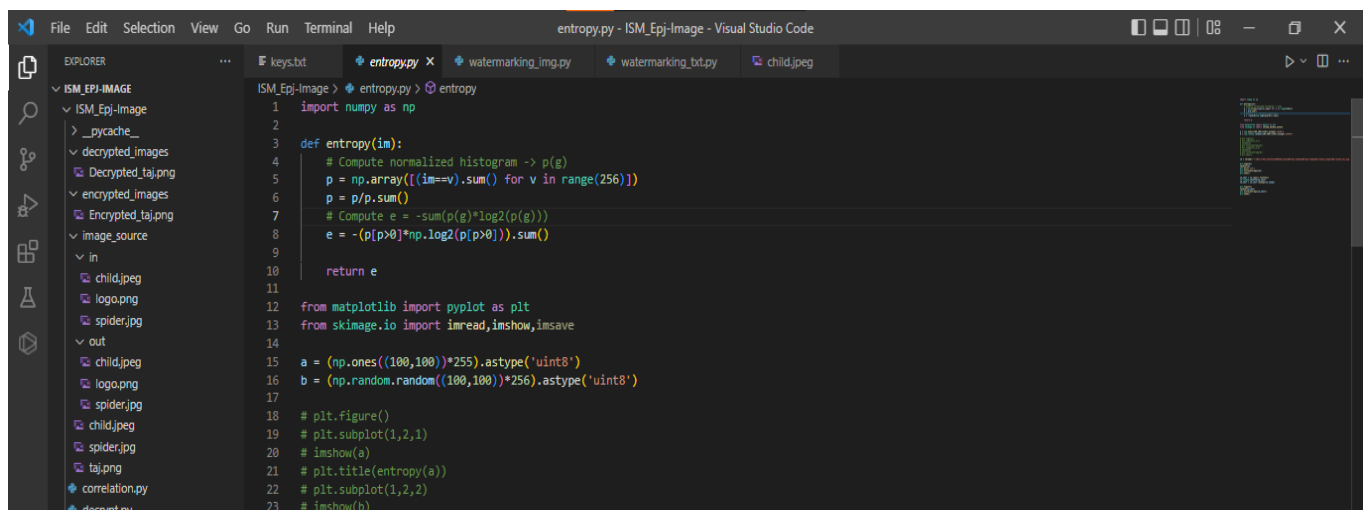
```

```

PS C:\Users\DELL\Desktop\ISM\ISM_Epj\ISM_Epj-Image> python -u "c:\Users\DELL\Desktop\ISM\ISM_Epj\ISM_Epj-Image\ISM_Epj-Image\correlation.py"
0.016904433122360535
PS C:\Users\DELL\Desktop\ISM\ISM_Epj\ISM_Epj-Image>
PS C:\Users\DELL\Desktop\ISM\ISM_Epj\ISM_Epj-Image>
PS C:\Users\DELL\Desktop\ISM\ISM_Epj\ISM_Epj-Image>

```

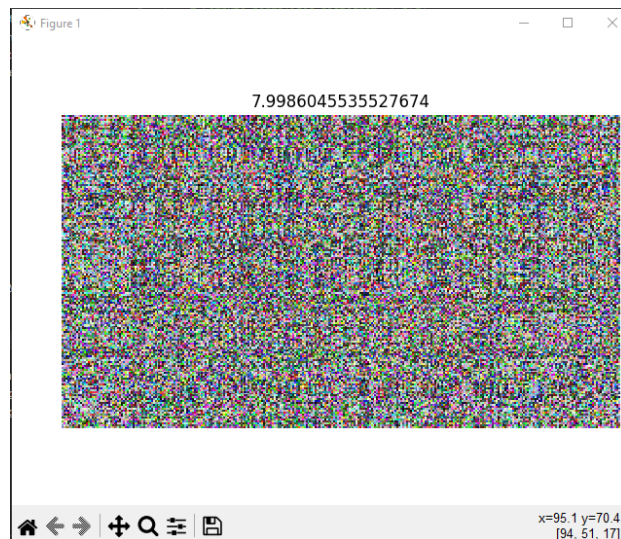
- Entropy of an image:



```

1 import numpy as np
2
3 def entropy(im):
4     # Compute normalized histogram -> p(g)
5     p = np.array([(im==v).sum() for v in range(256)])
6     p = p/p.sum()
7     # Compute e = -sum(p(g)*log2(p(g)))
8     e = -(p[p>0]*np.log2(p[p>0])).sum()
9
10    return e
11
12 from matplotlib import pyplot as plt
13 from skimage.io import imread,imshow,imsave
14
15 a = (np.ones((100,100))*255).astype('uint8')
16 b = (np.random.random((100,100))*256).astype('uint8')
17
18 # plt.figure()
19 # plt.subplot(1,2,1)
20 # imshow(a)
21 # plt.title(entropy(a))
22 # plt.subplot(1,2,2)
23 # imshow(b)

```

• Code Snippet for the Web Application:

➤ Runserver.py

```
➤ """  
➤ This script runs the Ism application using a development server.  
➤ """
```

```
➤ from os import environ  
➤ from Ism import app
```

```
➤ if __name__ == '__main__':  
➤ HOST = environ.get('SERVER_HOST', 'localhost')  
➤ try:  
➤ PORT = int(environ.get('SERVER_PORT', '5555'))  
➤ except ValueError:  
➤ PORT = 5555  
➤ app.run(HOST, PORT)
```

➤ Views.py

```
➤ """  
➤ Routes and views for the flask application.  
➤ """  
  
➤ from datetime import datetime  
➤ from flask import render_template  
➤ from Ism import app  
➤ from flask import request  
➤ import random  
➤ import os  
➤ from flask import Flask, flash, request, redirect, url_for
```

```

➤ from flask import send_from_directory
➤ from flask import send_file
➤ from PIL import Image
➤ from random import randint
➤ import numpy
➤ import sys
➤ from helper import *
➤ from PIL import Image
➤ from random import randint
➤ import numpy
➤ import sys
➤ from helper import *
➤ from PIL import Image
➤ from math import sqrt
➤ import cv2 as cv2

➤ Kr = []
➤ Kc = []

➤ def encrypt(file):
➤     global Kr, Kc
➤     # im = Image.open(r"C:\Users\DELL\Desktop\ISM\Ism_Epj\Image-Security-master\taj.png")
➤     im = Image.open(
➤         r"C:\Users\DELL\Desktop\ISM\Ism_Epj_Final\{f}".format(f=file))
➤     pix = im.load()
➤     # Obtaining the RGB matrices
➤     r = []
➤     g = []
➤     b = []
➤     for i in range(im.size[0]):
➤         r.append([])
➤         g.append([])
➤         b.append([])
➤         for j in range(im.size[1]):
➤             rgbPerPixel = pix[i, j]
➤             r[i].append(rgbPerPixel[0])
➤             g[i].append(rgbPerPixel[1])
➤             b[i].append(rgbPerPixel[2])

➤     m = im.size[0]
➤     n = im.size[1]

➤     # Vectors Kr and Kc
➤     alpha = 8
➤     Kr = [randint(0, pow(2, alpha)-1) for i in range(m)]
➤     Kc = [randint(0, pow(2, alpha)-1) for i in range(n)]
➤     ITER_MAX = 1

```

```

➤ print('Vector Kr : ', Kr)
➤ print('Vector Kc : ', Kc)

➤ f = open('keys.txt', 'w+')
➤ f.write('Vector Kr : \n')
➤ for a in Kr:
➤     f.write(str(a) + '\n')
➤ f.write('Vector Kc : \n')
➤ for a in Kc:
➤     f.write(str(a) + '\n')
➤ f.write('ITER_MAX : \n')
➤ f.write(str(ITER_MAX) + '\n')

➤ for iterations in range(ITER_MAX):
➤     # For each row
➤     for i in range(m):
➤         rTotalSum = sum(r[i])
➤         gTotalSum = sum(g[i])
➤         bTotalSum = sum(b[i])
➤         rModulus = rTotalSum % 2
➤         gModulus = gTotalSum % 2
➤         bModulus = bTotalSum % 2
➤         if(rModulus == 0):
➤             r[i] = numpy.roll(r[i], Kr[i])
➤         else:
➤             r[i] = numpy.roll(r[i], -Kr[i])
➤         if(gModulus == 0):
➤             g[i] = numpy.roll(g[i], Kr[i])
➤         else:
➤             g[i] = numpy.roll(g[i], -Kr[i])
➤         if(bModulus == 0):
➤             b[i] = numpy.roll(b[i], Kr[i])
➤         else:
➤             b[i] = numpy.roll(b[i], -Kr[i])
➤     # For each column
➤     for i in range(n):
➤         rTotalSum = 0
➤         gTotalSum = 0
➤         bTotalSum = 0
➤         for j in range(m):
➤             rTotalSum += r[j][i]
➤             gTotalSum += g[j][i]
➤             bTotalSum += b[j][i]
➤         rModulus = rTotalSum % 2
➤         gModulus = gTotalSum % 2
➤         bModulus = bTotalSum % 2
➤         if(rModulus == 0):
➤             upshift(r, i, Kc[i])
➤         else:

```

```

➤ downshift(r, i, Kc[i])
➤ if(gModulus == 0):
➤ upshift(g, i, Kc[i])
➤ else:
➤ downshift(g, i, Kc[i])
➤ if(bModulus == 0):
➤ upshift(b, i, Kc[i])
➤ else:
➤ downshift(b, i, Kc[i])
➤ # For each row
➤ for i in range(m):
➤ for j in range(n):
➤ if(i % 2 == 1):
➤ r[i][j] = r[i][j] ^ Kc[j]
➤ g[i][j] = g[i][j] ^ Kc[j]
➤ b[i][j] = b[i][j] ^ Kc[j]
➤ else:
➤ r[i][j] = r[i][j] ^ rotate180(Kc[j])
➤ g[i][j] = g[i][j] ^ rotate180(Kc[j])
➤ b[i][j] = b[i][j] ^ rotate180(Kc[j])
➤ # For each column
➤ for j in range(n):
➤ for i in range(m):
➤ if(j % 2 == 0):
➤ r[i][j] = r[i][j] ^ Kr[i]
➤ g[i][j] = g[i][j] ^ Kr[i]
➤ b[i][j] = b[i][j] ^ Kr[i]
➤ else:
➤ r[i][j] = r[i][j] ^ rotate180(Kr[i])
➤ g[i][j] = g[i][j] ^ rotate180(Kr[i])
➤ b[i][j] = b[i][j] ^ rotate180(Kr[i])

➤ for i in range(m):
➤ for j in range(n):
➤ pix[i, j] = (r[i][j], g[i][j], b[i][j])

➤ im.save(r"C:\Users\DELL\Desktop\ISM\IsM_Epj_Final\Image-Security-
master\IsM\static\content\Encrypted.png")
➤ return (Kr, r"C:\Users\DELL\Desktop\ISM\IsM_Epj_Final\Image-Security-
master\IsM\static\content\Encrypted.png")
➤ # fo=open("enc.jpg","wb")
➤ # imageRes="enc.jpg"
➤ # fo.write(image)
➤ # fo.close()
➤ # return (key,imageRes)

➤ def decrypt(key, file):
➤ global Kr, Kc

```

```

> im = Image.open(
> r"C:\Users\DELL\Desktop\ISM\Ism_Epj_Final\Image-Security-master\Ism\static\content\Encrypted.png")
> pix = im.load()

> # Obtaining the RGB matrices
> r = []
> g = []
> b = []
> for i in range(im.size[0]):
> r.append([])
> g.append([])
> b.append([])
> for j in range(im.size[1]):
> rgbPerPixel = pix[i, j]
> r[i].append(rgbPerPixel[0])
> g[i].append(rgbPerPixel[1])
> b[i].append(rgbPerPixel[2])

> m = im.size[0]
> n = im.size[1]

> # Kr = []
> # Kc = []

> # print('Enter value of Kr')

> # for i in range(m):
> #     Kr.append(int(input()))

> # print('Enter value of Kc')
> # for i in range(n):
> #     Kc.append(int(input()))

> # print('Enter value of ITER_MAX')
> # ITER_MAX = int(input())

> for iterations in range(1):
> # For each column
> for j in range(n):
> for i in range(m):
> if(j % 2 == 0):
> r[i][j] = r[i][j] ^ Kr[i]
> g[i][j] = g[i][j] ^ Kr[i]
> b[i][j] = b[i][j] ^ Kr[i]
> else:
> r[i][j] = r[i][j] ^ rotate180(Kr[i])

```

```

➤ g[i][j] = g[i][j] ^ rotate180(Kr[i])
➤ b[i][j] = b[i][j] ^ rotate180(Kr[i])
➤ # For each row
➤ for i in range(m):
➤ for j in range(n):
➤ if(i % 2 == 1):
➤ r[i][j] = r[i][j] ^ Kc[j]
➤ g[i][j] = g[i][j] ^ Kc[j]
➤ b[i][j] = b[i][j] ^ Kc[j]
➤ else:
➤ r[i][j] = r[i][j] ^ rotate180(Kc[j])
➤ g[i][j] = g[i][j] ^ rotate180(Kc[j])
➤ b[i][j] = b[i][j] ^ rotate180(Kc[j])
➤ # For each column
➤ for i in range(n):
➤ rTotalSum = 0
➤ gTotalSum = 0
➤ bTotalSum = 0
➤ for j in range(m):
➤ rTotalSum += r[j][i]
➤ gTotalSum += g[j][i]
➤ bTotalSum += b[j][i]
➤ rModulus = rTotalSum % 2
➤ gModulus = gTotalSum % 2
➤ bModulus = bTotalSum % 2
➤ if(rModulus == 0):
➤ downshift(r, i, Kc[i])
➤ else:
➤ upshift(r, i, Kc[i])
➤ if(gModulus == 0):
➤ downshift(g, i, Kc[i])
➤ else:
➤ upshift(g, i, Kc[i])
➤ if(bModulus == 0):
➤ downshift(b, i, Kc[i])
➤ else:
➤ upshift(b, i, Kc[i])

➤ # For each row
➤ for i in range(m):
➤ rTotalSum = sum(r[i])
➤ gTotalSum = sum(g[i])
➤ bTotalSum = sum(b[i])
➤ rModulus = rTotalSum % 2
➤ gModulus = gTotalSum % 2
➤ bModulus = bTotalSum % 2
➤ if(rModulus == 0):
➤ r[i] = numpy.roll(r[i], -Kr[i])
➤ else:
➤ r[i] = numpy.roll(r[i], Kr[i])

```

```

➤ if(gModulus == 0):
➤     g[i] = numpy.roll(g[i], -Kr[i])
➤ else:
➤     g[i] = numpy.roll(g[i], Kr[i])
➤ if(bModulus == 0):
➤     b[i] = numpy.roll(b[i], -Kr[i])
➤ else:
➤     b[i] = numpy.roll(b[i], Kr[i])

➤ for i in range(m):
➤     for j in range(n):
➤         pix[i, j] = (r[i][j], g[i][j], b[i][j])

➤ im.save(r"C:\Users\DELL\Desktop\ISM\Ism_Epj_Final\Image-Security-
➤ master\Ism\static\content\Decrypted.png")
➤ return r"C:\Users\DELL\Desktop\ISM\Ism_Epj_Final\Image-Security-
➤ master\Ism\static\content\Decrypted.png"

➤ # fo=open("dec.jpg","wb")
➤ # imageRes="dec.jpg"
➤ # fo.write(image)
➤ # fo.close()
➤ # return imageRes

➤ def correlation():

➤     value_of_x = 0
➤     value_of_y = 0

➤     # color_index_of_rgb 0-red,1-green,2-blue

➤     def co1(color_index_of_rgb, height, width, pixels):
➤         value = 0
➤         for pixel_coordinate_of_y in range(0, height):
➤             for pixel_coordinate_of_x in range(0, width):
➤                 if pixel_coordinate_of_x+1 == width:
➤                     break
➤                 value = pixels[pixel_coordinate_of_x, pixel_coordinate_of_y][color_index_of_rgb] * \
➤                 pixels[pixel_coordinate_of_x+1,
➤                 pixel_coordinate_of_y][color_index_of_rgb]+value

➤         return value*height*width

➤     def co2(color_index_of_rgb, height, width, pixels):
➤         global value_of_y
➤         global value_of_x
➤         for pixel_coordinate_of_y in range(0, height):

```

```

➤ for pixel_coordinate_of_x in range(0, width):
➤ if pixel_coordinate_of_x+1 == width:
➤ break
➤ value_of_x = pixels[pixel_coordinate_of_x,
➤ pixel_coordinate_of_y][color_index_of_rgb]+value_of_x
➤ value_of_y = pixels[pixel_coordinate_of_x+1,
➤ pixel_coordinate_of_y][color_index_of_rgb]+value_of_y

➤ return value_of_x*value_of_y

➤ def co3(color_index_of_rgb, height, width, pixels):
➤ value = 0
➤ for pixel_coordinate_of_y in range(0, height):
➤ for pixel_coordinate_of_x in range(0, width):
➤ value = (pixels[pixel_coordinate_of_x,
➤ pixel_coordinate_of_y][color_index_of_rgb])**2 + value

➤ xy = (value*height*width)-(value_of_x**2)
➤ return xy

➤ def co4(color_index_of_rgb, height, width, pixels):
➤ value = 0
➤ for pixel_coordinate_of_y in range(0, height):
➤ for pixel_coordinate_of_x in range(0, width):
➤ if pixel_coordinate_of_x+1 == width:
➤ break
➤ value = (pixels[pixel_coordinate_of_x+1,
➤ pixel_coordinate_of_y][color_index_of_rgb]**2)+value

➤ xy = (value*height*width)-(value_of_y**2)
➤ return xy

➤ def corr_of_rgb(loc):
➤ global value_of_y
➤ global value_of_x
➤ photo = Image.open(loc)
➤ # cryptotiger.bmp
➤ pixels = photo.load()
➤ width, height = photo.size
➤ value_of_y = 0
➤ value_of_x = 0
➤ r = ((co1(0, height, width, pixels)-co2(0, height, width, pixels)) /
➤ sqrt(co3(0, height, width, pixels)*co4(0, height, width, pixels)))
➤ value_of_y = 0
➤ value_of_x = 0
➤ g = ((co1(1, height, width, pixels) - co2(1, height, width, pixels)) /
➤ sqrt(co3(1, height, width, pixels) * co4(1, height, width, pixels)))

```



```

➤ value_of_x = 0
➤ value_of_y = 0
➤ b = ((co1(2, height, width, pixels) - co2(2, height, width, pixels)) /
➤ sqrt(co3(2, height, width, pixels) * co4(2, height, width, pixels)))

➤ ans = (r+g+b)/3
➤ return ans

➤ enc = corr_of_rgb(
➤ r"C:\Users\DELL\Desktop\ISM\Ism_Epj_Final\Image-Security-master\Ism\static\content\Encrypted.png")
➤ dnc = corr_of_rgb(
➤ r"C:\Users\DELL\Desktop\ISM\Ism_Epj_Final\Image-Security-master\Ism\static\content\Decrypted.png")
➤ return (enc, dnc)

➤ def Entropy():
➤ def entropy(im):
➤ # Compute normalized histogram -> p(g)
➤ p = numpy.array([(im == v).sum() for v in range(256)])
➤ p = p/p.sum()
➤ # Compute e = -sum(p(g)*log2(p(g)))
➤ e = -(p[p > 0]*numpy.log2(p[p > 0])).sum()

➤ return e

➤ enc=entropy(cv2.imread(
➤ r'C:\Users\DELL\Desktop\ISM\Ism_Epj_Final\Image-Security-master\Ism\static\content\Encrypted.png'))
➤ dnc=entropy(cv2.imread(
➤ r'C:\Users\DELL\Desktop\ISM\Ism_Epj_Final\Image-Security-master\Ism\static\content\Decrypted.png'))
➤ return (enc,dnc)

➤ @app.route('/')
➤ @app.route('/home')
➤ def home():
➤ """Renders the home page."""
➤ return render_template(
➤ 'index.html',
➤ title='Home Page',
➤ year=datetime.now().year,
➤ )

➤ @app.route('/contact')
➤ def contact():
➤ """Renders the contact page."""
➤ return render_template(
➤ 'contact.html',
➤ title='Decrypt',

```

```

➤ year=datetime.now().year,
➤ message='Upload your encrypted image along with the key'
➤ )

➤ @app.route('/about')
➤ def about():
➤     """Renders the about page."""
➤     return render_template(
➤         'about.html',
➤         title='Encrypt',
➤         year=datetime.now().year,
➤         message='Upload the image here'
➤     )

➤ @app.route('/contact1', methods=['POST'])
➤ def contact1():
➤     if request.method == 'POST':
➤         global f
➤         f = request.files['file']
➤         f.save(f.filename)
➤         text = request.form['key']
➤         key = int(text)
➤         image = decrypt(key, f.filename)
➤         return render_template('contact1.html',
➤             title='Decrypted',
➤             year=datetime.now().year,
➤             message='This is your Decrypted image', name=f.filename)

➤ @app.route('/about1', methods=['POST'])
➤ def about1():
➤     if request.method == 'POST':
➤         global f
➤         f = request.files['file']
➤         f.save(f.filename)
➤         key, image = encrypt(f.filename)
➤         return render_template('about1.html',
➤             title='Encrypted',
➤             year=datetime.now().year,
➤             message='This is your encrypted image', name=f.filename, keys=key, images=image)

➤ @app.route('/cor', methods=['GET'])
➤ def cor():
➤     if request.method == 'GET':
➤         global f
➤         #f = request.files['file']
➤         # f.save(f.filename)
➤         enc, dnc = correlation()

```

```
➤ return render_template('cor.html',
➤ title='Correlation',
➤ year=datetime.now().year,
➤ message='This is your correlation values of Image', keys=enc, images=dnc)

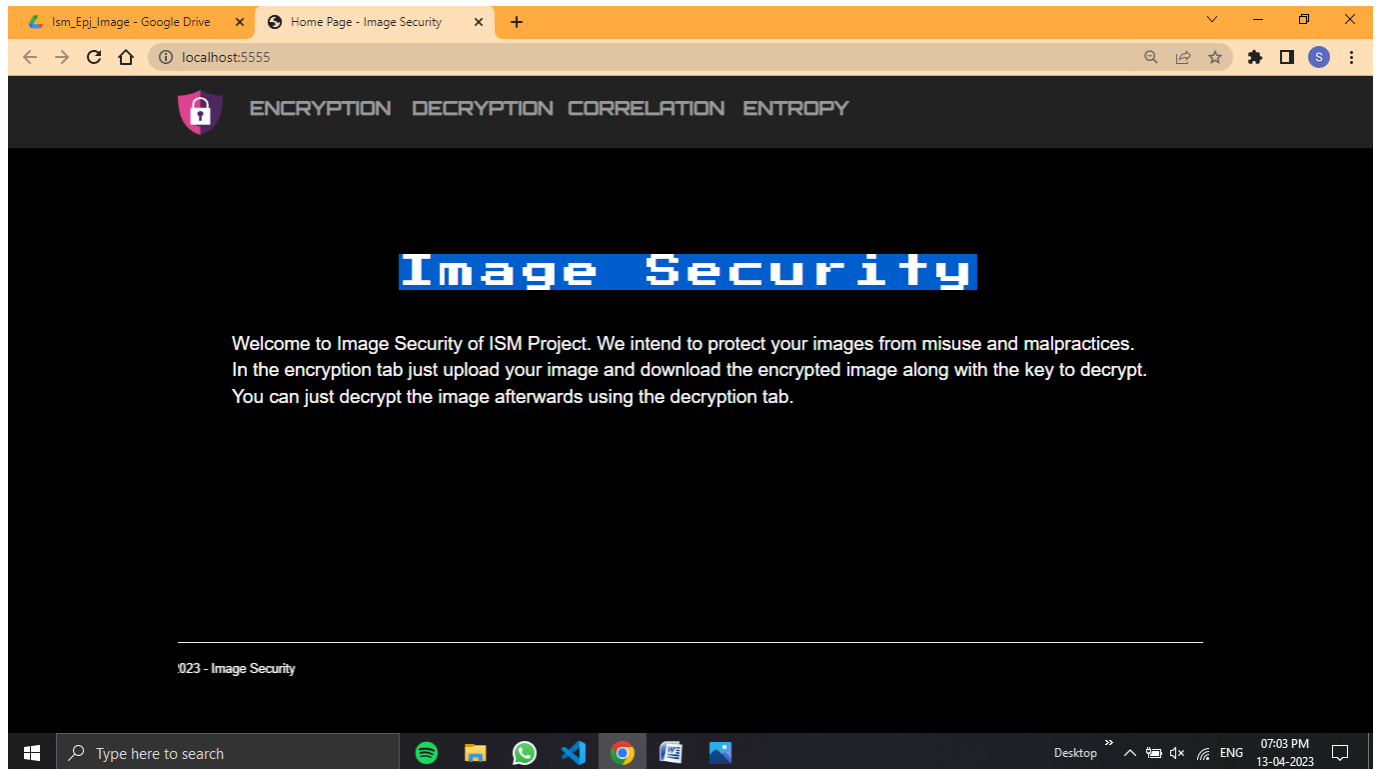
➤ @app.route('/ent', methods=['GET'])
➤ def ent():
➤     if request.method == 'GET':
➤         global f
➤         #f = request.files['file']
➤         # f.save(f.filename)
➤         enc, dnc = Entropy()
➤         return render_template('ent.html',
➤ title='Entropy',
➤ year=datetime.now().year,
➤ message='This is your Entropy values of Image', keys=enc, images=dnc)

➤ @app.route('/return-file')
➤ def return_file():
➤     return send_file("../enc.jpg", attachment_filename="enc.jpg")

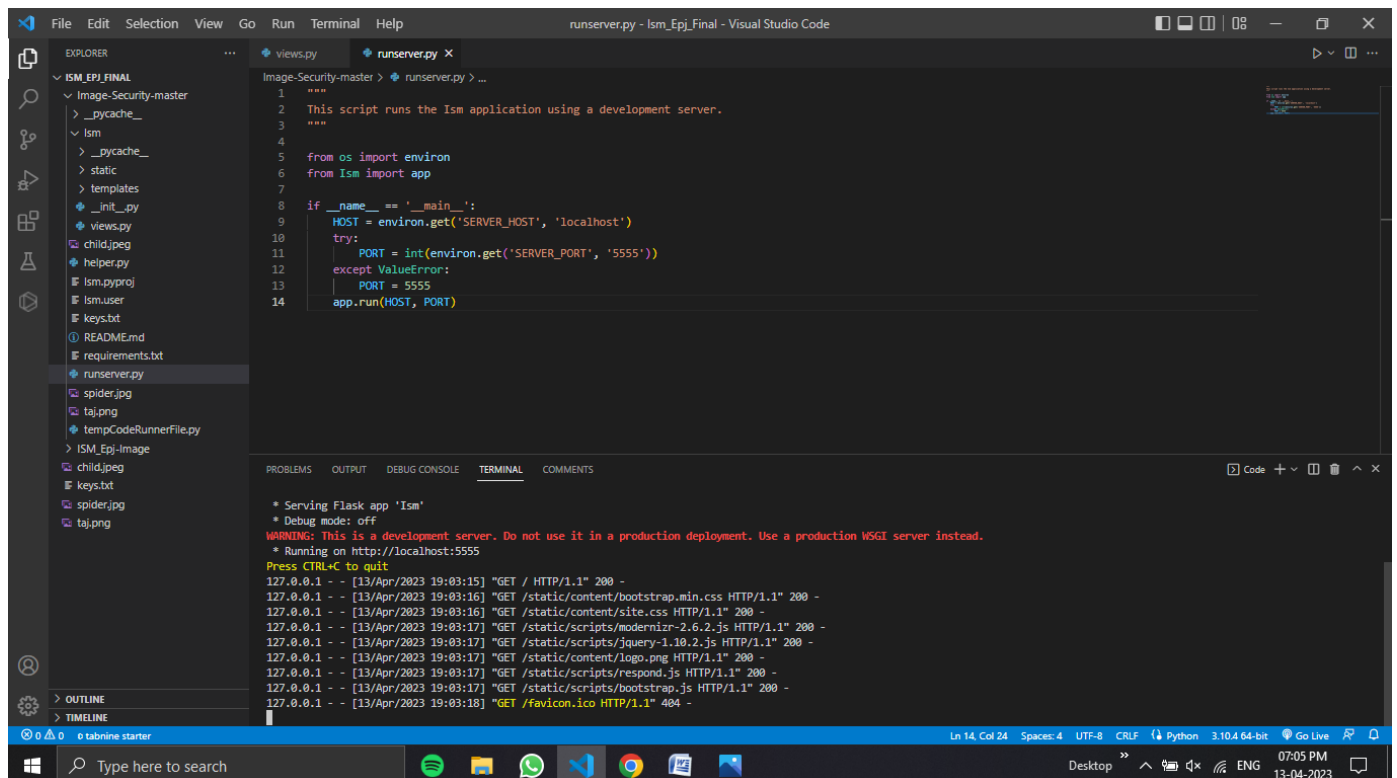
➤ @app.route('/return-file1')
➤ def return_file1():
➤     return send_file("../dec.jpg", attachment_filename="dec.jpg")
```

- **Web Application of Image Security:**

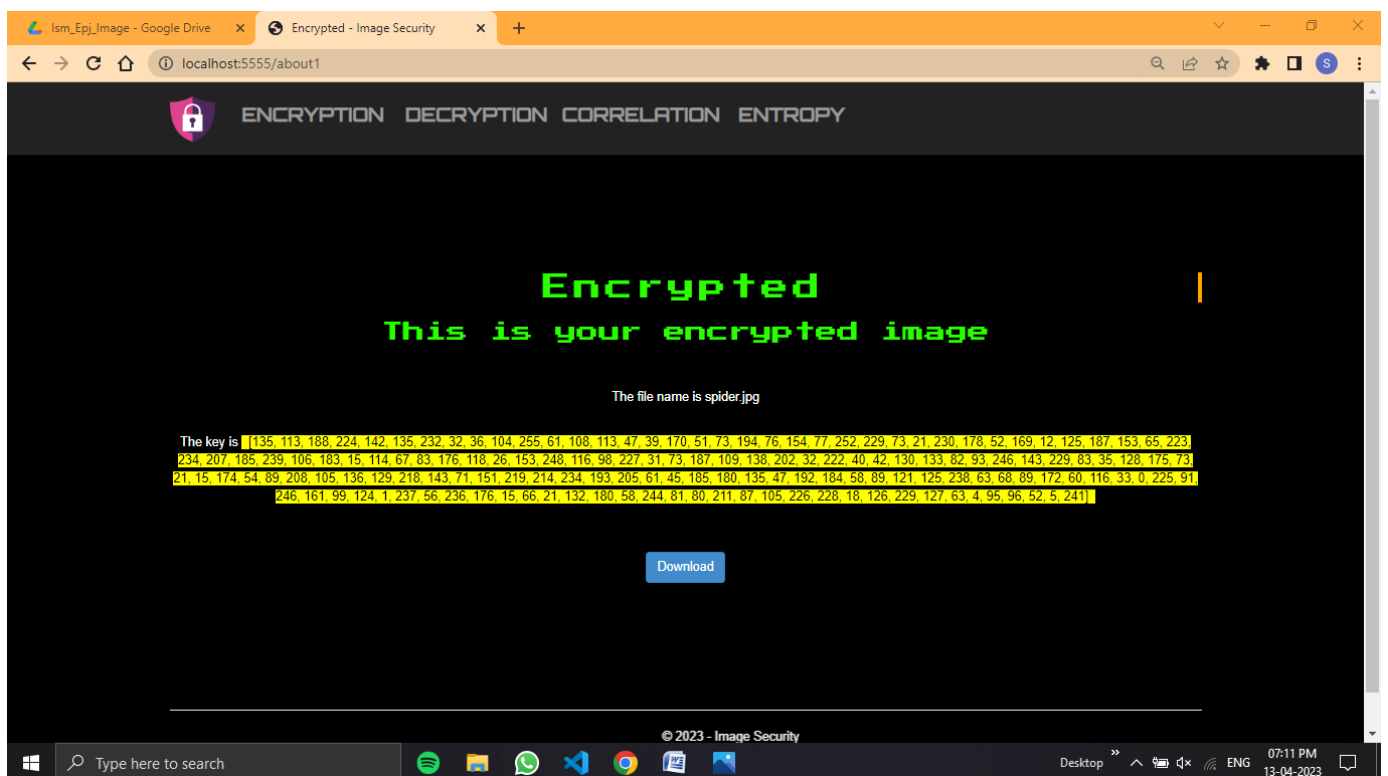
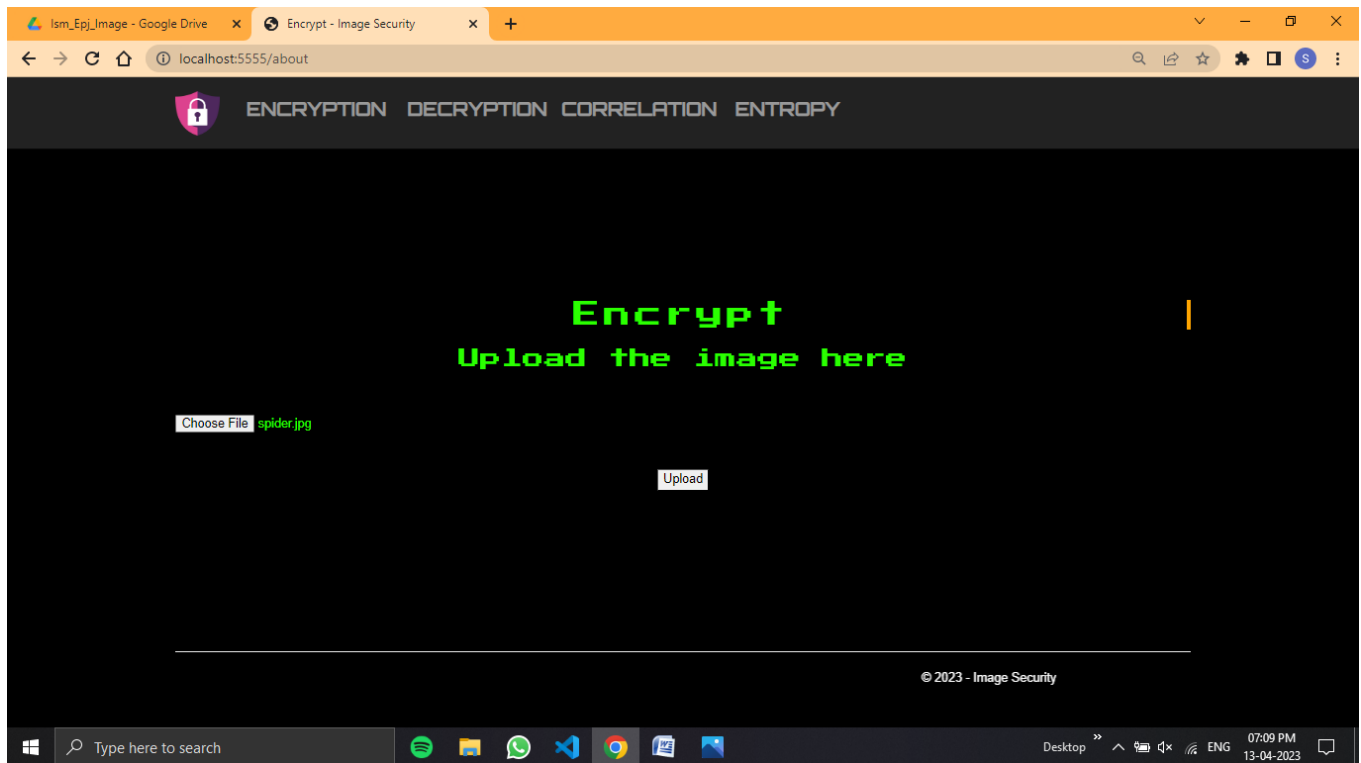
- **Home Page**



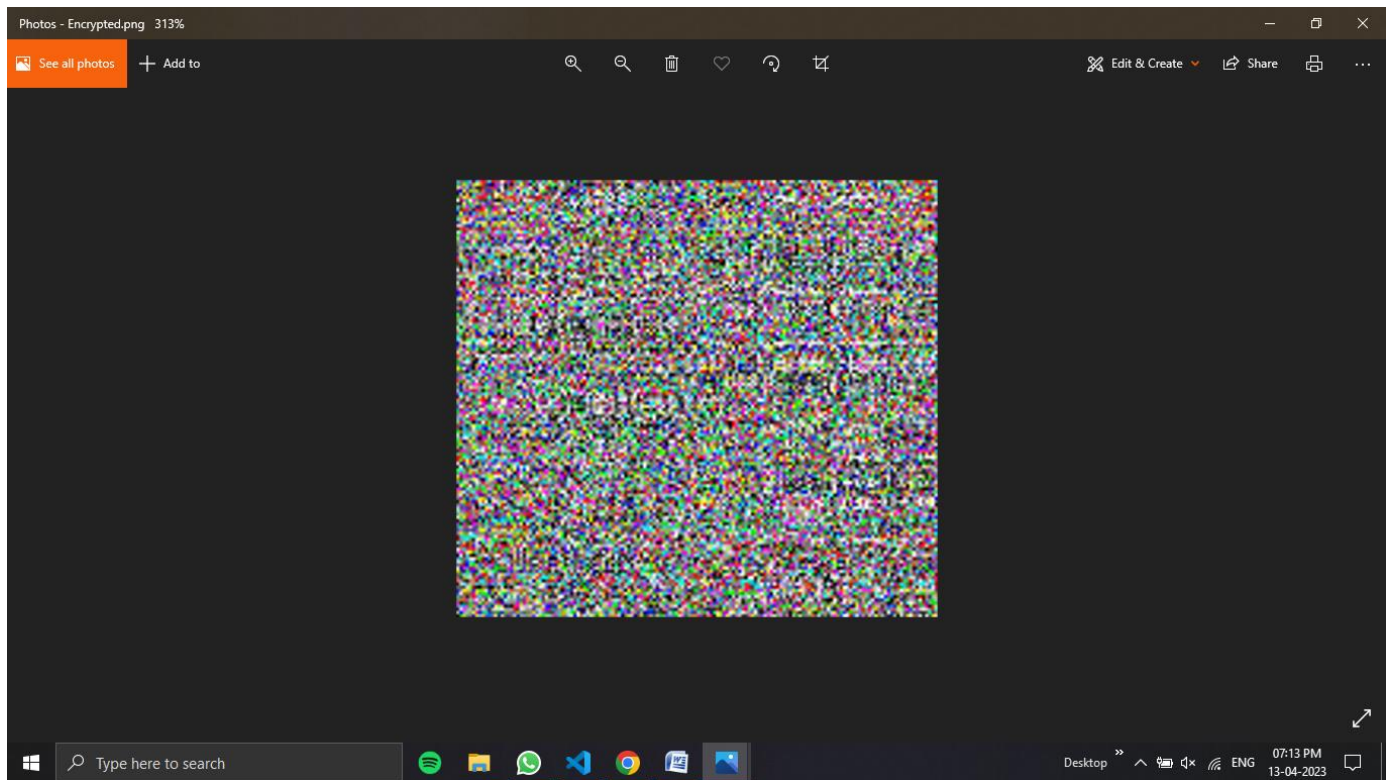
- **Run Server.py The Main Code for the Web Application.**



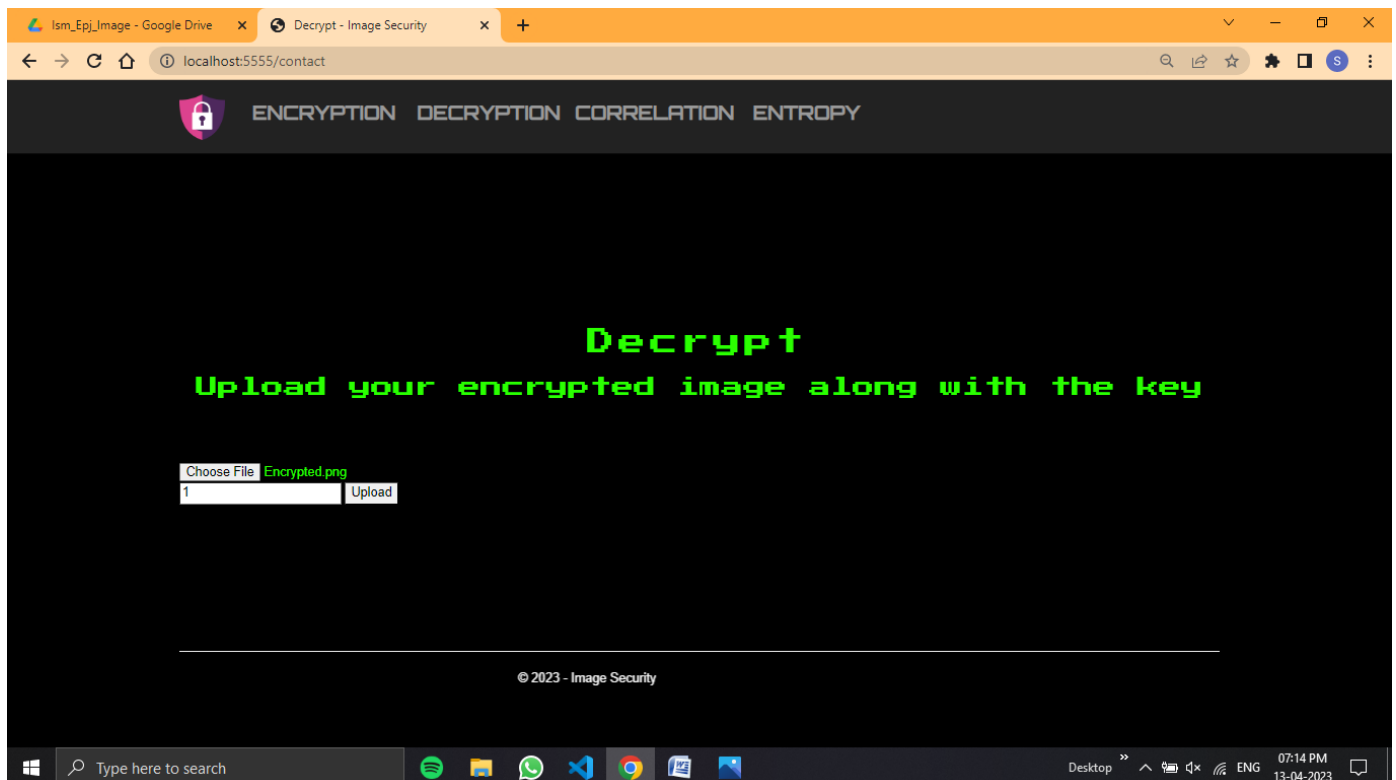
- Encryption of the Image.



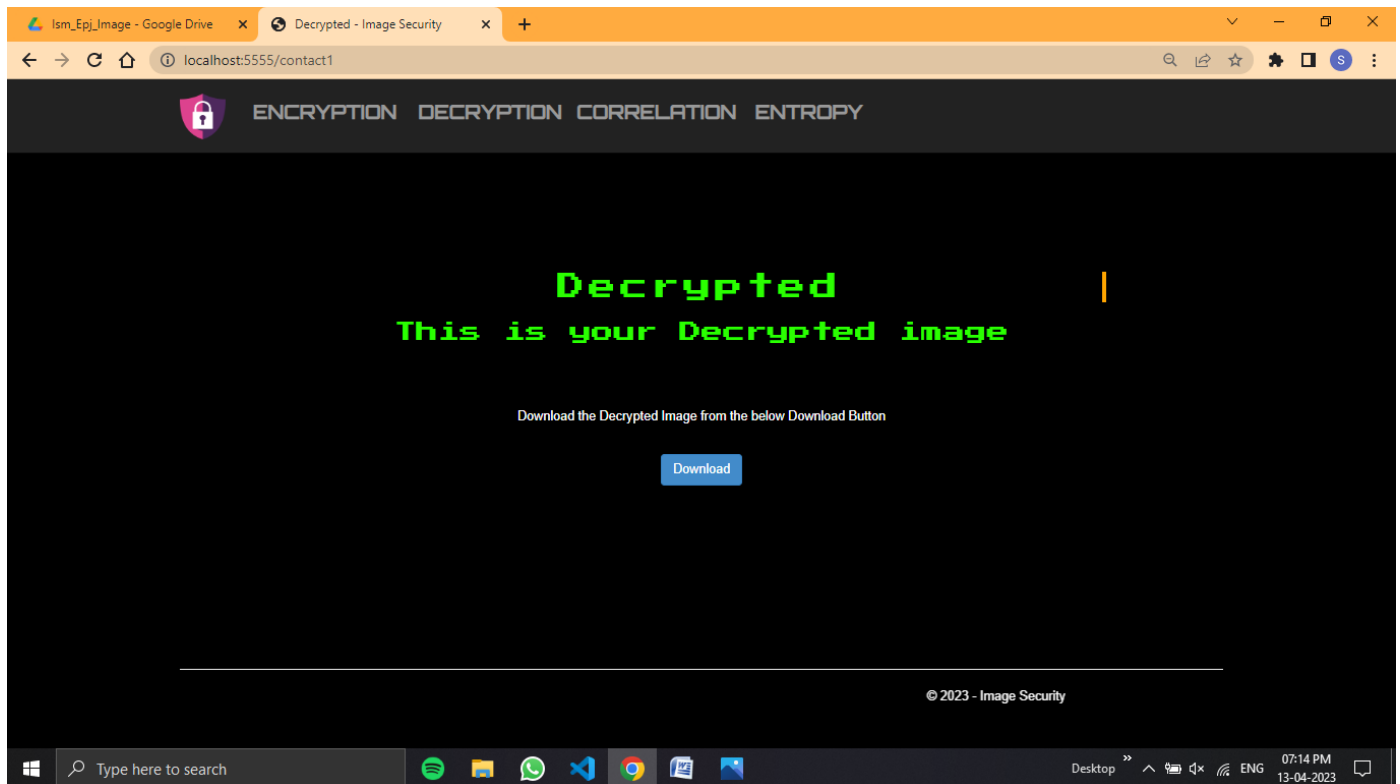
- **Download the image and save the encrypted Image:**



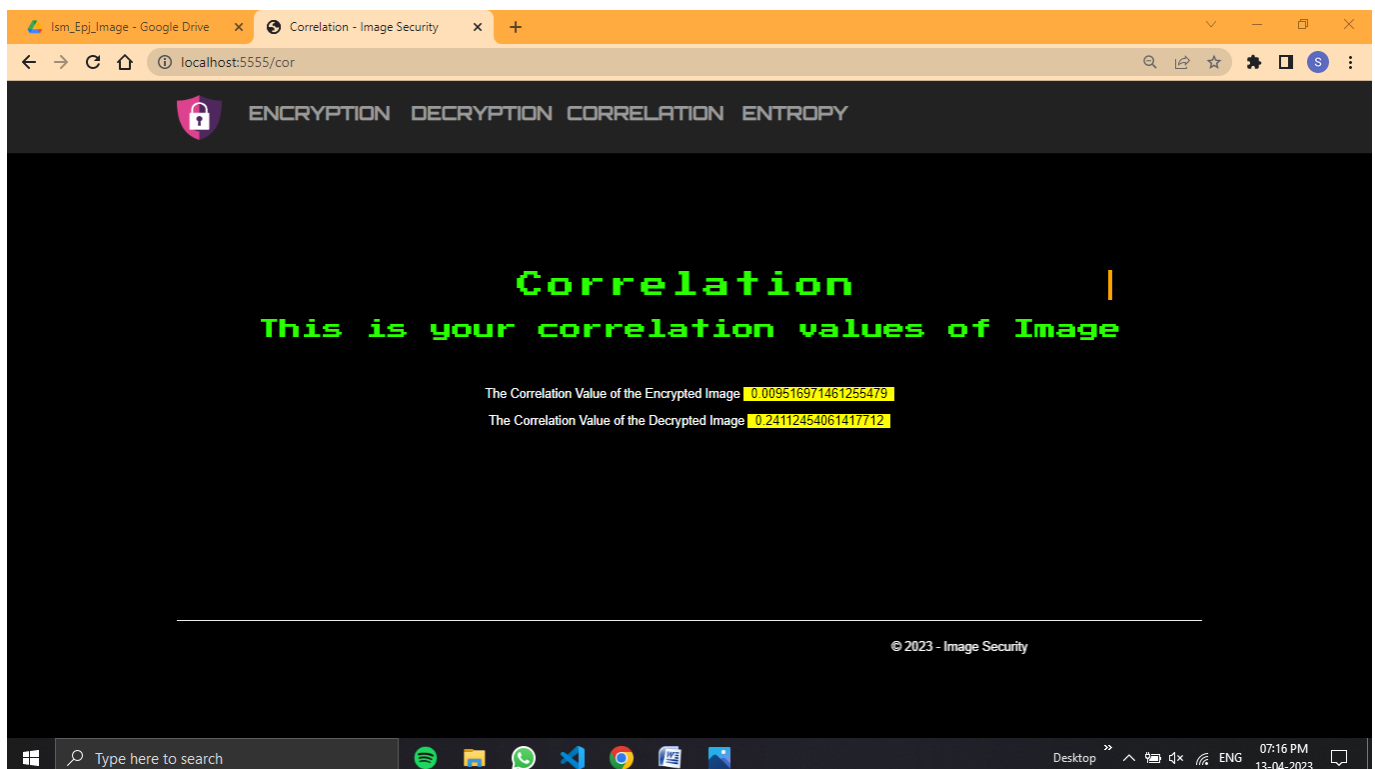
- **Decryption of the Image Screenshot:**



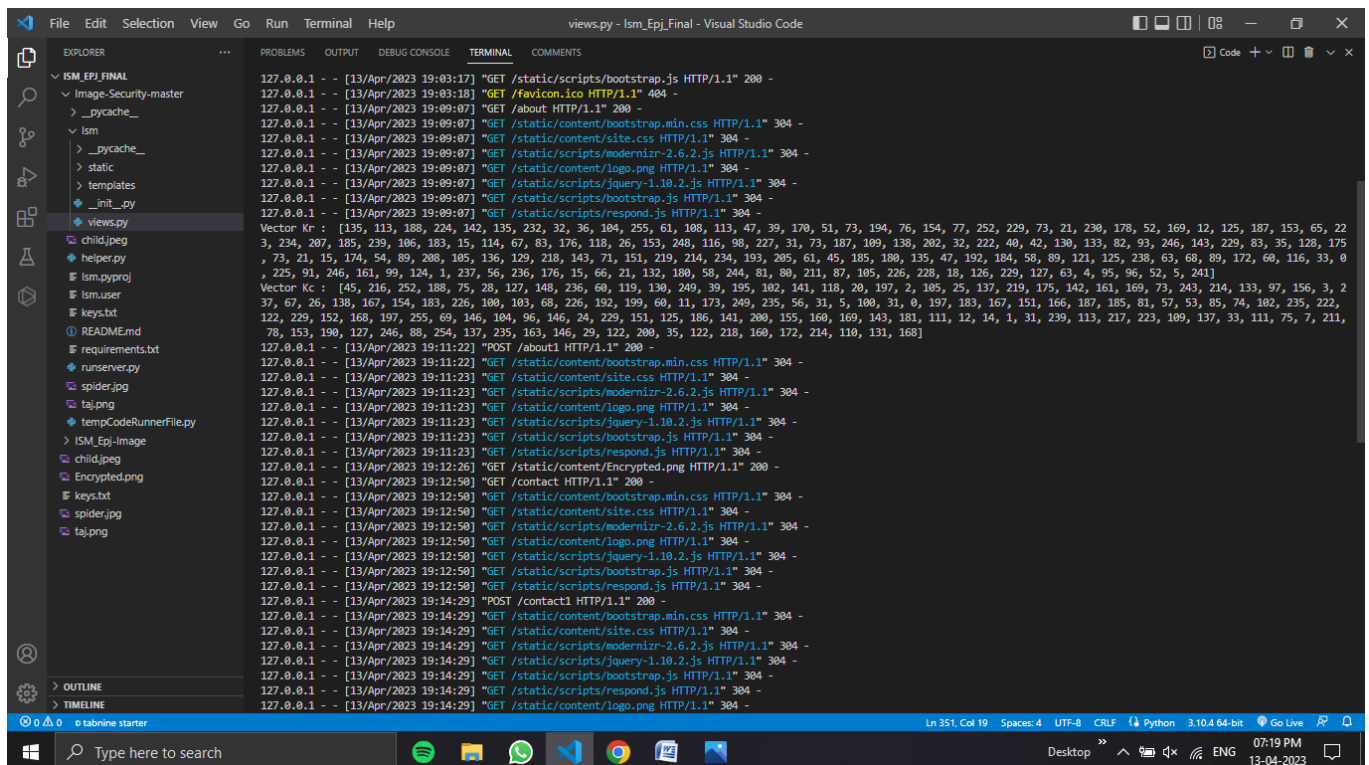
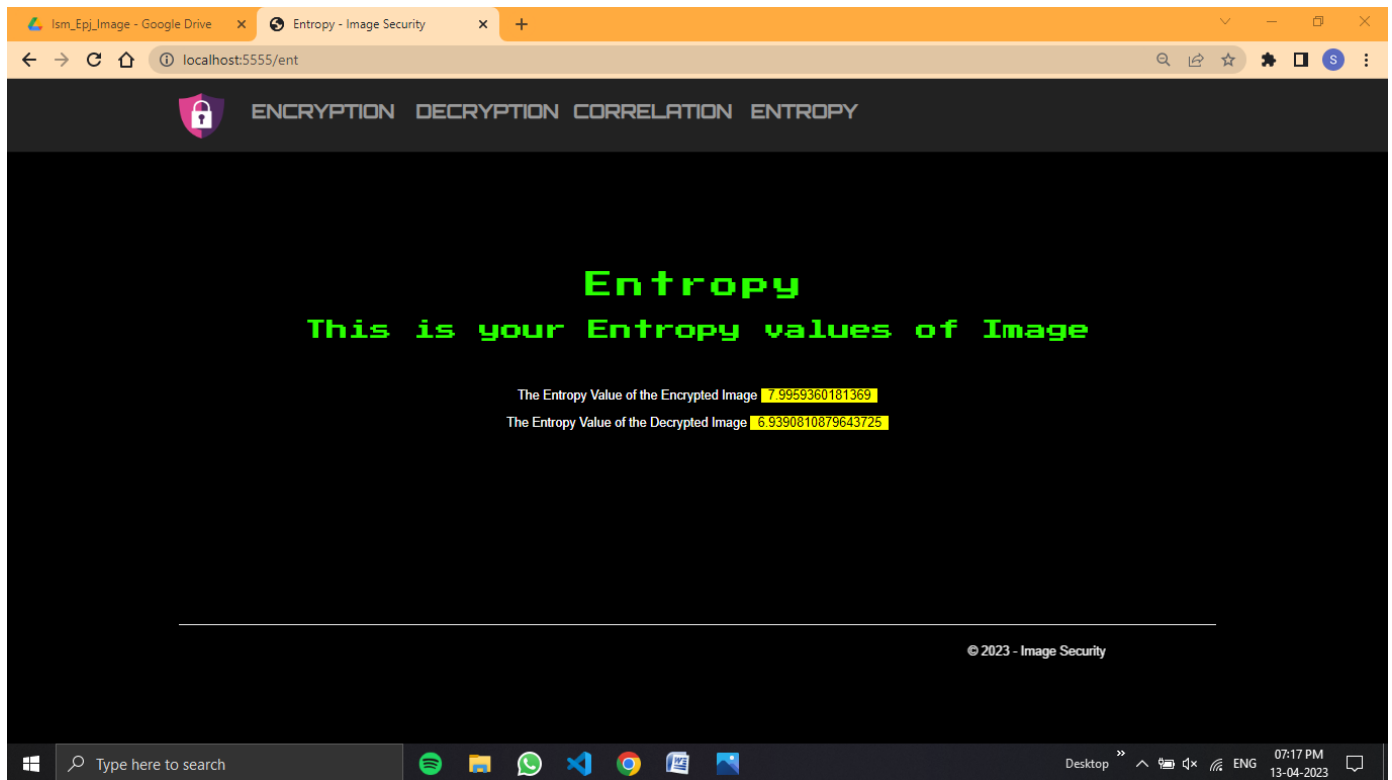
- **Download the image and save the Decrypted Image:**



- **Correlation Values before and after the encryption:**



- **Entropy Values before and after the encryption:**



Conclusion:

Image Name	Correlation of Encrypted Image	Correlation of Decrypted Image
Spider.png	0.0095	0.2411
Child.jpg	0.072	0.31
Taj.jpeg	0.013	0.15

Image Name	Entropy of Encrypted Image	Entropy of Decrypted Image
Spider.png	7.99	6.93
Child.jpg	5.65	4.25
Taj.jpeg	6.34	5.18

The Rubik's Cube principle and image watermarking are effective methods for enhancing image security. By using the principle of the Rubik's Cube to shuffle the pixels of an image, it becomes difficult for unauthorized users to decipher the original image. Additionally, watermarking allows for the embedding of a unique identifier within an image, making it easier to track and identify unauthorized copies of the image.

In this project, the effectiveness of these methods was evaluated by comparing the correlation and entropy values of the original image with those of the modified images. The results showed that the Rubik's Cube shuffling and watermarking both led to significant changes in both correlation and entropy values, indicating an improvement in image security.

Overall, the combination of the Rubik's Cube principle and image watermarking provides a powerful solution for image security, making it difficult for unauthorized users to tamper with or copy images. However, it is important to note that no security method is foolproof, and it is always important to employ multiple layers of security to protect sensitive information.

References:

- [1] T. G. Research and B. L. Shivakumar, "Multistage Image Encryption using Rubik's Cube for Secured Image Transmission," *International Journal of Advanced Research in Computer Science*, vol. 6, no. 6, pp. 54–58, 2015, doi: 10.26483/IJARCS.V6I6.2544.
- [2] V. M. Ionescu and A. V. Diaconu, "Rubik's cube principle based image encryption algorithm implementation on mobile devices," *Proceedings of the 2015 7th International Conference on Electronics, Computers and Artificial Intelligence, ECAI 2015*, pp. 31–34, Oct. 2015, doi: 10.1109/ECAI.2015.7301247.
- [3] R. Vidhya and M. Brindha, "A chaos based image encryption algorithm using Rubik's cube and prime factorization process (CIERPF)," *Journal of King Saud University - Computer and Information Sciences*, Jan. 2020, doi: 10.1016/J.JKSUCI.2019.12.014.
- [4] M. Helmy, E. S. M. El-Rabaie, I. M. Eldokany, and F. E. A. El-Samie, "3-D Image Encryption Based on Rubik's Cube and RC6 Algorithm," *3D Research*, vol. 8, no. 4, Dec. 2017, doi: 10.1007/S13319-017-0145-8.
- [5] N. K. Pareek, V. Patidar, and K. K. Sud, "Diffusion–substitution based gray image encryption scheme," *Digital Signal Processing*, vol. 23, no. 3, pp. 894–901, May 2013, doi: 10.1016/J.DSP.2013.01.005.
- [6] B. Mondal and T. Mandal, "A light weight secure image encryption scheme based on chaos & DNA computing," *Journal of King Saud University - Computer and Information Sciences*, vol. 29, no. 4, pp. 499–504, Oct. 2017, doi: 10.1016/J.JKSUCI.2016.02.003.
- [7] X. Wang, L. Teng, and X. Qin, "A novel colour image encryption algorithm based on chaos," *Signal Processing*, vol. 92, no. 4, pp. 1101–1108, Apr. 2012, doi: 10.1016/J.SIGPRO.2011.10.023.
- [8] Q. Lu, C. Zhu, and X. Deng, "An Efficient Image Encryption Scheme Based on the LSS Chaotic Map and Single S-Box," *IEEE Access*, vol. 8, pp. 25664–25678, 2020, doi: 10.1109/ACCESS.2020.2970806.
- [9] K. Loukhaoukha, J. Y. Chouinard, and A. Berdai, "A secure image encryption algorithm based on Rubik's cube principle," *Journal of Electrical and Computer Engineering*, 2012, doi: 10.1155/2012/173931.
- [10] V. M. Ionescu and A. V. Diaconu, "Rubik's cube principle based image encryption algorithm implementation on mobile devices," *Proceedings of the 2015 7th International Conference on Electronics, Computers and Artificial Intelligence, ECAI 2015*, pp. 31–34, Oct. 2015, doi: 10.1109/ECAI.2015.7301247.

- [11] X. Qian, Q. Yang, Q. Li, Q. Liu, Y. Wu, and W. Wang, “A Novel Color Image Encryption Algorithm Based on Three-Dimensional Chaotic Maps and Reconstruction Techniques,” IEEE Access, vol. 9, pp. 61334–61345, 2021, doi: 10.1109/ACCESS.2021.3073514.
- [12] K. A. Abitha and P. K. Bharathan, “Secure Communication Based on Rubik’s Cube Algorithm and Chaotic Baker Map,” Procedia Technology, vol. 24, pp. 782–789, 2016, doi: 10.1016/J.PROTCY.2016.05.089.
- [13] W. Huang, D. Jiang, Y. An, L. Liu, and X. Wang, “A Novel Double-Image Encryption Algorithm Based on Rossler Hyperchaotic System and Compressive Sensing,” IEEE Access, vol. 9, pp. 41704–41716, 2021, doi: 10.1109/ACCESS.2021.3065453.
- [14] L. Liu, D. Jiang, T. An, and Y. Guan, “A Plaintext-Related Dynamical Image Encryption Algorithm Based on Permutation-Combination-Diffusion Architecture,” IEEE Access, vol. 8, pp. 62785–62799, 2020, doi: 10.1109/ACCESS.2020.2983716.
- [15] W. Huang, D. Jiang, Y. An, L. Liu, and X. Wang, “A Novel Double-Image Encryption Algorithm Based on Rossler Hyper chaotic System and Compressive Sensing,” IEEE Access, vol. 9, pp. 41704–41716, 2021, doi: 10.1109/ACCESS.2021.3065453.

- **Drive Link for All the Output and the Code Snippets:**

[Link](#)