A

MINI PROJECT REPORT

ON

IMAGE ENCRYPTION USING CENTRAL DOGMA

Submitted in partial fulfillment of the requirements

For the award of Degree of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

Submitted By

Bale Sujith Kumar	**245321733068**
Chilla Sravan	**245321733074**
Gaini Santhosh Kumar	**245321733080**
Pattaparla Varun Goud	**245321733115**

Under the guidance

Of

Mrs. S. Meghana

ASSISTANT PROFESSOR

Department of CSE

NEIL GOGTE INSTITUTE OF TECHNOLOGY

Kachavanisingaram Village, Hyderabad, Telangana 500058.

NEIL GOGTE INSTITUTE OF TECHNOLOGY

A Unit of Keshav Memorial Technical Education (KMTES)

Approved by AICTE, New Delhi & Affiliated to Osmania University, Hyderabad

CERTIFICATE

This is to certify that the Mini project work entitled "**IMAGE ENCRYPTION USING CENTRAL DOGMA**" *is a bonafide work carried out by* **Bale SujithKumar(245321733068) , Chilla Sravan(245321733074) , Gaini SanthoshKumar(245321733080) , Pattaparla Varun Goud(245321733115)** of III-year VI semester **Bachelor of Engineering** *in CSE by Osmania University, Hyderabad during the academic year* **2023-2024** *is a record of bonafide work carried out by them. The results embodied in this report have not been submitted to any other University or Institution for the award of any degree.*

Internal Guide

Mrs. S .Meghana

Assistant Professor

Head of Department

Dr. K V Ranga Rao

Professor

External

DECLARATION

I hereby declare that the Mini Project Report entitled, "**IMAGE ENCRYPTION USING CENTRAL DOGMA**" submitted for the B.E degree is entirely my work and all ideas and references have been duly acknowledged. It does not contain any work for the award of any other degree.

Date:

B.Sujith Kumar	**245321733068**
Chilla Sravan	**245321733074**
G.Santosh Kumar	**245321733080**
P.Varun Goud	**245321733068**

ACKNOWLEDGEMENT

I am happy to express our deep sense of gratitude to the principal of the college **Dr . R. Shyam Sunder, Professor**, Neil Gogte Institute of Technology, for having provided us with adequate facilities to pursue our project.

I would like to thank **Dr . K V Ranga Rao**, **Head of the Department, CSE Department**, And **Mini Project Coordinator Dr . B Krishna Associate Professor CSE Department**, Neil Gogte Institute of Technology, for having provided the freedom to use all the facilities available in the department, especially the laboratories and the library.

I would also like to thank my internal guide **Mrs. S. Meghana Assistant Professor CSE Department**, for her technical guidance & constant encouragement.

I sincerely thank my seniors and all the teaching and non-teaching staff of the Department of Computer Science & Engineering for their timely suggestions, healthy criticism and motivation during this work.

Finally, I express my immense gratitude with pleasure to the other individuals who have either directly or indirectly contributed to our need at the right time for the development and success of this work.

ABSTRACT

In the era of burgeoning data exchange and storage, ensuring the confidentiality and integrity of sensitive information is paramount. This proposes a novel multiple image encryption and decryption algorithm that leverages the fundamental principles of the central dogma of molecular biology, offering a unique paradigm for secure data transformation. By drawing inspiration from the flow of genetic information from DNA to RNA to protein, our algorithm orchestrates a sequential process to encrypt and decrypt images efficiently while maintaining robust security measures.

Furthermore, chaotic systems are seamlessly integrated into the encryption process to provide an additional layer of security, augmenting the resilience against unauthorized access and attacks. Chaotic systems introduce unpredictability and complexity, thereby enhancing the cryptographic strength of the algorithm.

The integration of central dogma principles and chaotic systems not only ensures enhanced security but also fosters computational efficiency, making our algorithm suitable for real-world applications where secure data transformation is imperative.

TABLE OF CONTENTS

List of Figures

List of Tables

CHAPTER – 1 INTRODUCTION

1.1 PROBLEM STATEMENT

The existing encryption algorithms often face challenges in providing both efficient data transformation and robust security measures for image encryption. Additionally, conventional encryption methods may lack the ability to adequately protect sensitive information from emerging cryptographic attacks. Hence, there is a pressing need to develop a novel image encryption and decryption algorithm that addresses these limitations by leveraging innovative approaches such as the integration of central dogma principles and chaotic systems. This algorithm should ensure efficient data transformation while enhancing overall data protection against unauthorized access and attacks.

1.2 MOTIVATION

The motivation behind this project stems from the increasing reliance on digital data, particularly images, for communication, storage, and transmission across various domains including healthcare, finance, and security. As the volume of digital data continues to grow, so does the need for robust encryption techniques to safeguard sensitive information from unauthorized access and cyber threats. Conventional encryption methods, while effective to some extent, may not offer adequate protection against sophisticated attacks.

By integrating principles from the central dogma of molecular biology and chaotic systems into image encryption, this project seeks to offer a novel approach that not only enhances the security of sensitive information but also ensures efficient data transformation. The utilization of central dogma principles provides a structured framework for data transformation, mimicking the flow of genetic information in biological systems.

Additionally, the incorporation of chaotic systems introduces unpredictability and complexity, thereby enhancing the overall security of the encryption process.

1.3 SCOPE

The scope of this to develop a novel image encryption and decryption algorithm integrating central dogma principles and chaotic systems. The algorithm will be designed to efficiently encrypt and decrypt images while enhancing data protection against unauthorized access and attacks. It will provide a robust framework for secure data transformation, catering to diverse applications across industries such as healthcare, finance, and security. This project aims to develop a novel image encryption and decryption algorithm integrating central dogma principles and chaotic systems. The algorithm will be designed to efficiently encrypt and decrypt multiple images while enhancing data protection against unauthorized access and attacks. It will provide a robust framework for secure data transformation, catering to diverse applications across industries such as healthcare, finance, and security.

1.4 OUTLINE

The project outline involves defining requirements, conducting a literature review, designing the algorithm, implementing it using a programming language like Python, and utilizing libraries like NumPy. The algorithm will incorporate principles from central dogma and chaotic systems for robust security and efficient data transformation. Through this approach, the project aims to develop a novel encryption and decryption solution for safeguarding sensitive image data effectively.

CHAPTER – 2 LITERATURE SURVEY

EXISTING SYSTEM:

Traditional encryption algorithms, such as AES and RSA, are commonly used for securing digital images during transmission. However, their effectiveness may diminish in the face of rapidly advancing technology, leaving image transfer vulnerable to unauthorized access. These algorithms, while widely employed, may not provide adequate protection against sophisticated decryption techniques, necessitating the exploration of more robust solutions.

PROPOSED SYSTEM:

Recent advancements in chaotic cryptography, inspired by the rapid evolution of chaos theory, present a promising avenue for enhancing image security. By leveraging chaotic systems in conjunction with central dogma principles, encryption algorithms can significantly augment security measures and increase complexity, rendering decryption challenging for unauthorized users. This innovative approach underscores the importance of adopting advanced encryption methodologies to fortify image security and address vulnerabilities in digital image transmission, ensuring confidentiality and integrity across multimedia communication channels.

CHAPTER - 3

SOFTWARE REQUIREMENTS SPECIFICATION

3.1 <u>Overall Description:</u>

This SRS provides a comprehensive overview of the project on image encryption and decryption algorithm project aims to develop a novel solution leveraging principles from central dogma and chaotic systems. It will involve designing and implementing an algorithm using Python, with support from libraries like NumPy. The solution will ensure robust security and efficient data transformation for safeguarding sensitive image data. Thorough testing and validation will be conducted to verify the algorithm's effectiveness and reliability.

3.2. <u>Operating Environment:</u>

The operating environment for the image encryption and decryption algorithm project involves utilizing software systems conducive to cryptographic operations and user interface (UI) development. The algorithm will be designed to run on systems supporting Python programming language, preferably with versions compatible with libraries such as NumPy. Additionally, we will employ Streamlit, a Python library for creating interactive web-based UIs, to develop the user interface component of the application.

This project is platform-independent, capable of deployment on various operating systems including Windows, macOS, and Linux distributions. Streamlit, being a web-based framework, allows for the creation of intuitive and user-friendly interfaces accessible through web browsers, eliminating the need for platform-specific installations. Moreover, access to highspeed internet may be necessary for downloading and updating libraries and dependencies, as well as for deploying web-based applications.

3.3 Functional Requirements:

The algorithm must have the following features and functionalities

- Image Encryption: The system must be able to encrypt one or multiple images using the designed algorithm, leveraging principles from central dogma and chaotic systems.
- Image Decryption: It should provide functionality to decrypt encrypted images back to their original form, ensuring data integrity and accessibility.
- Error Handling: The system should handle errors gracefully, providing informative error messages to users in case of invalid inputs or unexpected failures during encryption or decryption processes.
- File Format Support: The system should support a variety of image file formats such as JPEG, PNG, and JPG for both input and output, ensuring compatibility with different types of image data.

3.4 Non-Functional Requirements:

Security: The algorithm must ensure robust security measures to protect sensitive image data from unauthorized access and cyber threats.

Performance: The system should exhibit high performance, with encryption and decryption processes being executed efficiently even for large image datasets.

Usability: The user interface should be intuitive and user-friendly, allowing users to interact with encryption and decryption functionalities without requiring specialized technical knowledge.

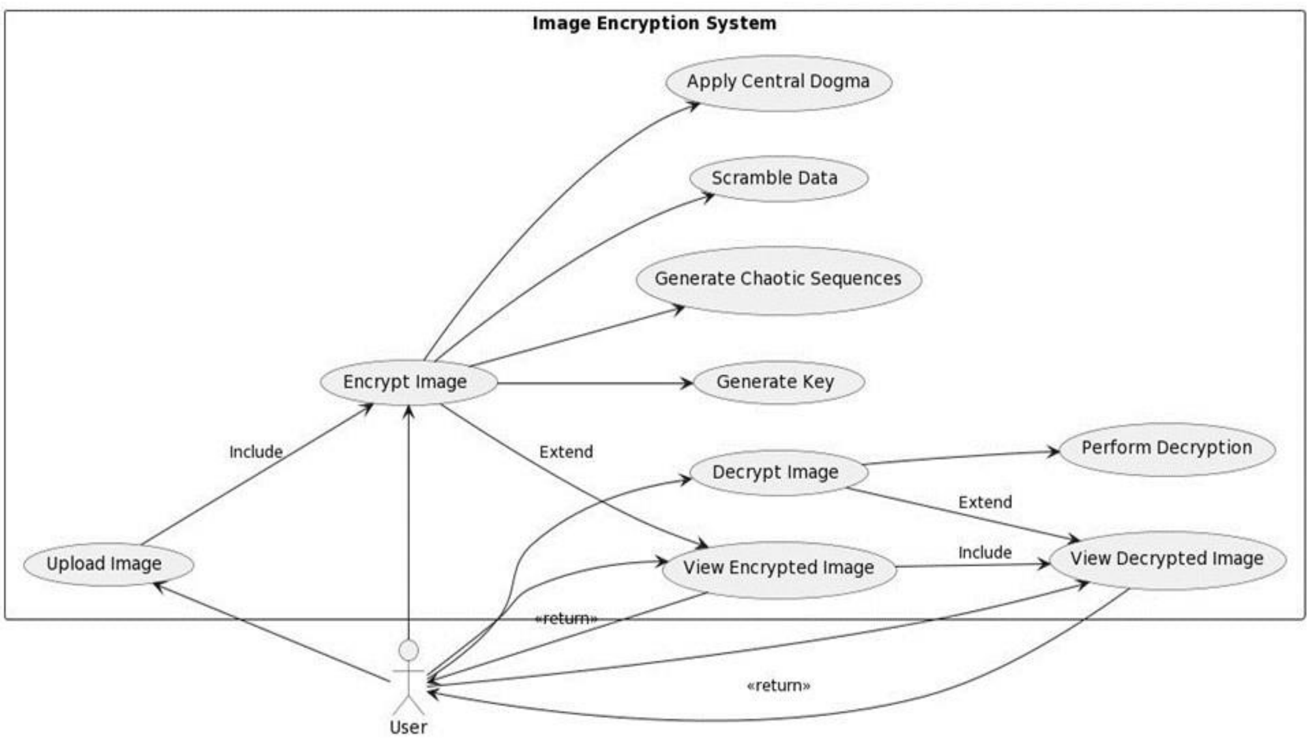
Reliability: The system must be reliable, ensuring that encryption and decryption processes are executed accurately and consistently under varying conditions.

Scalability: The solution should be scalable, capable of handling increasing loads of image encryption and decryption requests without compromising performance or security.

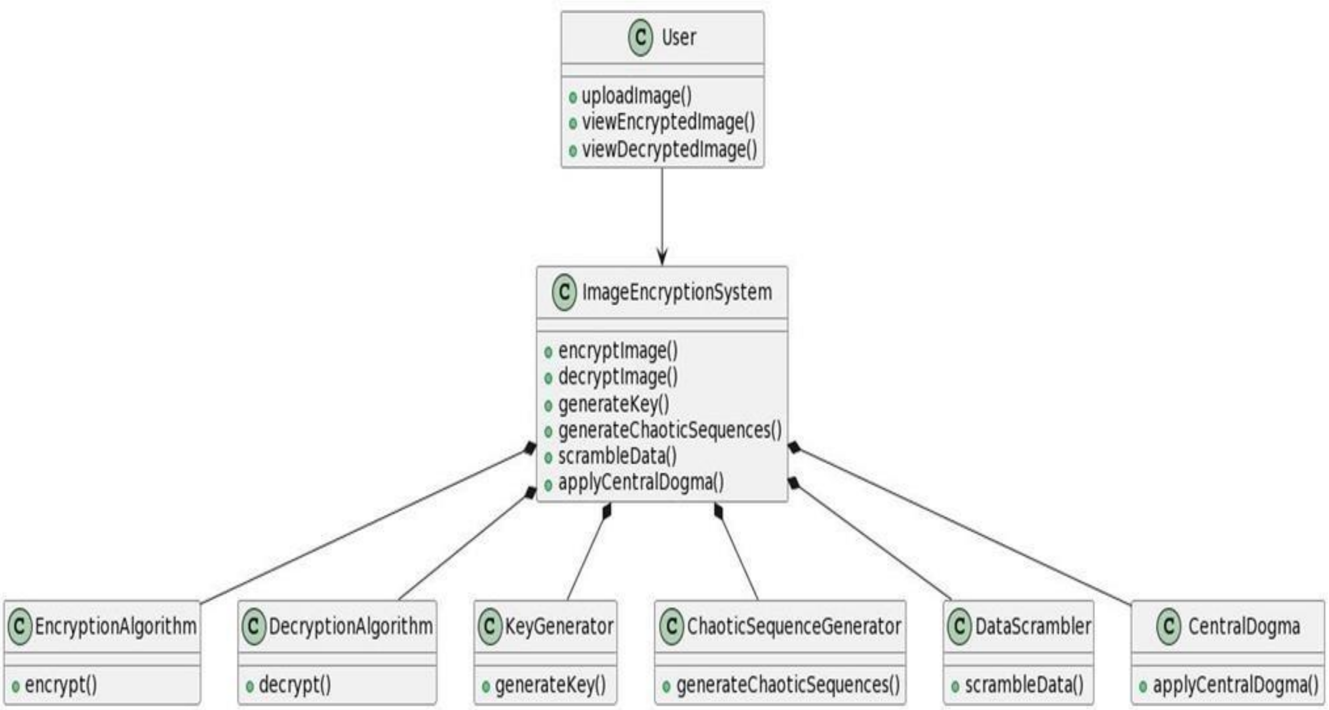
CHAPTER-4

SYSTEM DESIGN

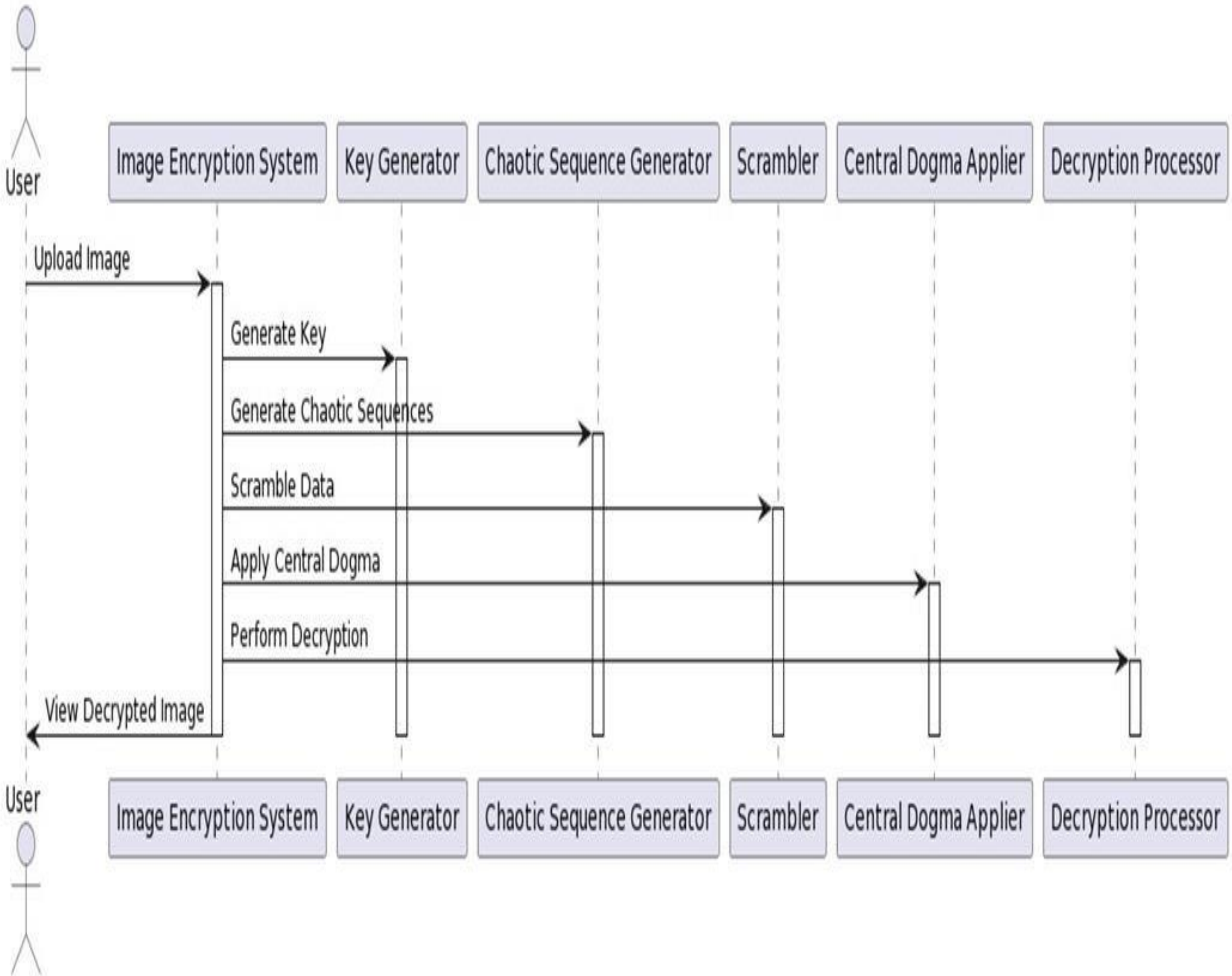
Use case Diagram:



CLASS DIAGRAM:



SEQUENCE DIAGRAM:



CHAPTER – 5 IMPLEMENTATION

5.1 SAMPLE CODE

```
import streamlit as st import
numpy as np from PIL import
Image import hashlib import
matplotlib.pyplot as plt


st.set_page_config(page_title="Geneix",page_icon=":lock_with_ink_pen:",layout:
"wide")
st.set_option('deprecation.showPyplotGlobalUse',
False) def main():
    st.title("Geneix Image Security")
st.subheader("Safeguarding")
st.title("Image  Encryption")        #
Upload multiple images
    uploaded_images = st.file_uploader("Upload Images", type=["jpg", "jpeg",     "png"],
accept_multiple_files=True)

    if uploaded_images:
        st.write(f"Number of Images Uploaded: {len(uploaded_images)}")
        # Display the uploaded images
```

```
 for i, img_file in enumerate(uploaded_images):

st.image(img_file, caption=f"Image {i+1}")

        # Encrypt button         if st.button("Encrypt

and Decrypt Images"):

            st.info("Encrypting...")

encrypted_images = []

decrypted_images = []

            # Perform encryption and decryption for each image

for i, img_file in enumerate(uploaded_images):

#st.write(f"\nEncrypting Image {i + 1}")                  img =

Image.open(img_file)  # Keep the image in RGB

                # Encryption                 image_hash, _ =

sha_256(img)  # Calculate SHA-256

                #st.info(f"SHA-256 for Image {i + 1}: {image_hash.hexdigest()}")

                encrypted_matrix,final_encrypted_matrix,rna_translation_matrix=
encryption_function(img)
encrypted_images.append(final_encrypted_matrix)

                # Decryption

                decrypted_matrix          =          decryption_function(img

      ,encrypted_matrix, rna_translation_matrix)

decrypted_images.append(decrypted_matrix)             st.success("Encryption and
```

Decryption complete!")   # Display the encrypted and decrypted images in

RGB

for i, (encrypted_img, decrypted_img) in enumerate(zip(encrypted_images, decrypted_images)):

plt.subplot(2, len(uploaded_images), i + 1)

plt.imshow(encrypted_img.astype(np.uint8))   plt.title(f'Encrypted

Image {i + 1}')   plt.axis("off")   plt.subplot(2,

len(uploaded_images), len(uploaded_images) + i + 1)

plt.imshow(decrypted_img.astype(np.uint8))   plt.title(f'Decrypted

Image {i + 1}')   plt.axis("off")   st.pyplot()

def sha_256(image):

Convert to numpy array and then to 8-bit binary

rgb_array = np.array(image)   # Generating SHA-

256   hash_sha256 = hashlib.sha256()

hash_sha256.update(rgb_array.tobytes())

hash_hexdigest = hash_sha256.hexdigest()

hash_values = [int(hash_hexdigest[i:i+2], 16) for i in

range(0, len(hash_hexdigest), 2)]   return hash_sha256, hash_values

external_keys = [1,2,3,4,5,6] def calculate_intermediate_params(c_values,

k_values):

Convert hexadecimal strings to integers

k_values = [int(k, 16) for k in k_values]

h1 = (c_values[0] + (k_values[0] ^ k_values[1] ^ k_values[2] ^ k_values[3] ^ k_values[4])) / 256

h2 = (c_values[1] + (k_values[5] ^ k_values[6] ^ k_values[7] ^ k_values[8] ^ k_values[9])) / 256

h3 = (c_values[2] + (k_values[10] ^ k_values[11] ^ k_values[12] ^ k_values[13] ^ k_values[14])) / 256

h4 = (c_values[3] + (k_values[15] ^ k_values[16] ^ k_values[17] ^ k_values[18] ^ k_values[19])) / 256

h5 = (c_values[4] + (k_values[20] ^ k_values[21] ^ k_values[22] ^ k_values[23] ^ k_values[24] ^ k_values[25])) / 256

h6 = (c_values[5] + (k_values[26] ^ k_values[27] ^ k_values[28] ^ k_values[29] ^ k_values[30] ^ k_values[31])) / 256   return h1,

h2, h3, h4, h5, h6 def calculate_initial_values(h1,

h2, h3, h4, h5, h6): x0 = ((h1 + h2 + h5) * 10**8)

% 256 / 255   y0 = ((h3 + h4 + h6) * 10**8) %

256 / 255   z0 = ((h1 + h2 + h3 + h4) * 10**8) %

256 / 255   p0 = ((h1 + h2 + h3) * 10**8) % 256

/ 255   q0 = ((h4 + h6 + h5) * 10**8) % 256 /

255   return x0, y0, z0, p0, q0 def

calculate_chaotic_system_parameters(h1, h2, h3,

h4, h5, h6):   a = (h1 + h2 / (h1 + h2 + h3 + h4

```
+ h5 + h6)) * 100 % 3 + 1     b = (h3 + h4 / (h1 +
h2 + h3 + h4 + h5 + h6)) * 100 % 3 + 1     c = (h5
+ h6 / (h1 + h2 + h3 + h4 + h5 + h6)) * 100 % 3 +
1     d = (h1 + h2 + h3 / (h1 + h2 + h3 + h4 + h5 +
h6)) % 1     return a, b, c, d
def dna_encoding_rules(rule_index):
    encoding_rules = {
        1: {'00': 'A', '11': 'T', '01': 'C', '10': 'G'},
        2: {'00': 'A', '11': 'T', '10': 'C', '01': 'G'},
        3: {'01': 'A', '10': 'T', '00': 'C', '11': 'G'},
        4: {'01': 'A', '10': 'T', '11': 'C', '00': 'G'},
        5: {'10': 'A', '01': 'T', '00': 'C', '11': 'G'},
        6: {'10': 'A', '01': 'T', '11': 'C', '00': 'G'},
        7: {'11': 'A', '00': 'T', '01': 'C', '10': 'G'},
8: {'11': 'A', '00': 'T', '10': 'C', '01': 'G'},
     }
    return encoding_rules[rule_index]   def
dna_encoding(p6):
  u, v, channels, w = p6.shape     encoded_dna = np.empty((u,
v, channels, w//2), dtype='U1')     for i in range(u):
```

```
    l_i=1         encoding_rule = dna_encoding_rules(l_i)         for j in
range(v):          for c in range(channels):             for k in range(0, w, 2):
value1 = p6[i, j, c, k]                 value2 = p6[i, j, c, k + 1]                 pair =
f"{value1}{value2}"  # Combine two values into one pair
encoded_dna[i, j, c, k // 2] = encoding_rule[pair]     return encoded_dna


transcription_table = {
    'A': 'U',
    'T': 'A',
    'C': 'G',
    'G':      'C'      }      def
dna_transcription(p7):
    # Apply the transcription rules using vectorization
p8      =      np.vectorize(transcription_table.get)(p7)
return p8 def generate_sequences_Y(p0, q0, d, u):
    Y = []    Z = []
p=p0    q=q0    for i
in range(u*4):
        p1 = np.sin(np.pi * d * (q + 3) * p * (1 - p))
q1 = np.sin(np.pi * d * (p + 3) * q * (1 - q))
Y.append(p1)        p = p1
```

```
        Z.append(q1)

        q = q1      return

Y                       def

mutation_rules():

    return {

        0: {'A': 'A', 'U': 'U', 'G': 'G', 'C': 'C'},

        1: {'A': 'U', 'U': 'A', 'G': 'C', 'C': 'G'},

        2: {'A': 'G', 'U': 'C', 'G': 'A', 'C': 'U'},

        3: {'A': 'C', 'U': 'G', 'G': 'U', 'C': 'A'}

    }

def rna_mutation(p8, Y):

    u, v, channels, w = p8.shape

    # Reshape Y to match the shape of p8

Y_reshaped = Y.reshape((u, v,channels, w))

p9 = np.empty_like(p8, dtype='U1')

    # Convert Y2 to integer Y1

    Y1 = np.floor(np.mod(Y_reshaped * 10**5, 4)).astype(int)

mutation_rules_dict = mutation_rules()      # Use vectorized

operations for efficient mutation

    mutation_func =        np.vectorize(lambda mode, base:

mutation_rules_dict[mode].get(base, base))     p9 = mutation_func(Y1, p8)
```

```
    return    p9         def
translation_rules():
    return {
        'A': 'U',
        'U': 'A',
        'C': 'G',
        'G': 'C'
    }
def rna_translation(p9):
    translation_rules_dict = translation_rules()     # Use vectorized operations
for  efficient  translation         translation_func  =  np.vectorize(lambda  base:
translation_rules_dict[base])     p10 = translation_func(p9)     return p10   def
generate_sequences_Z(p0, q0, d, u):
    Y = []
    Z1 = []
Z=[]     p=p0
q=q0     for i
in
range(u*4):
```

```
        p1 = np.sin(np.pi * d * (q + 3) * p * (1 - p))

q1 = np.sin(np.pi * d * (p + 3) * q * (1 - q))

Y.append(p1)        p = p1

        Z1.append(q1)

q  =  q1        for  i  in

range(u):

        Z.append(Z1[i])

return        Z                def

rna_encoding(Z, p10, u):

    # Transform Z into the range of 0-255

    Z_prime = np.floor(np.mod(Z * 10**5, 256)).astype(np.uint8)

    # Reshape Z_prime into a 3D array

    Z_reshaped = Z_prime.reshape(p10.shape[0], p10.shape[1],p10.shape[2])

    #       Create       bit       planes                              rna_bit_planes       =

np.unpackbits(np.expand_dims(Z_reshaped, axis=-1), axis=-1)

    # Reshape the bit planes to form a 4D array      rna_bit_planes_4d =

rna_bit_planes.reshape(Z_reshaped.shape + (8,))

    # Apply encoding rules

    encoding_rules = {

        '00': 'A',

        '11': 'U',
```

```
        '01': 'C',

        '10': 'G'

    }

   # Initialize a 4D array to store the RNA sequence with the desired shape

    rna_sequence_4d                  =                  np.empty((rna_bit_planes_4d.shape[0],
rna_bit_planes_4d.shape[1],rna_bit_planes_4d.shape[2], 4), dtype='U1')

    # Iterate through the 3D array      for i in

range(rna_bit_planes_4d.shape[0]):         for j in

range(rna_bit_planes_4d.shape[1]):             for c in

range(rna_bit_planes_4d.shape[2]):              # Pair 2-bits

and apply encoding rules                for k in range(0,

rna_bit_planes_4d.shape[3], 2):

                bit_pair  =  ''.join(map(str,  rna_bit_planes_4d[i,  j,  c,  k:k+2]))

rna_sequence_4d[i, j, c, k:k+2] = list(encoding_rules[bit_pair])      return

rna_sequence_4d xor_truth_table = {

    'A': {'A': 'A', 'U': 'U', 'C': 'C', 'G': 'G'},

    'U': {'A': 'U', 'U': 'A', 'C': 'G', 'G': 'C'},

    'C': {'A': 'C', 'U': 'G', 'C': 'A', 'G': 'U'},

    'G': {'A': 'G', 'U': 'C', 'C': 'U', 'G': 'A'}

}    def    rna_computing(encoded_array,

p10):
```

Assuming the first value of P11 is the same as the encoded array

P11 = np.zeros_like(p10, dtype=np.object_)

#P11[0] = encoded_array[0]

u = P11.shape[0] for j in range(0, u):

Vectorized XOR operation using the truth table

P11[j] = np.vectorize(lambda x, y: xor_truth_table[x][y])(encoded_array[j], p10[j]) return P11 def binary_matrix_to_decimal(matrix): # Reshape the matrix to a 3D array flattened_array = matrix.reshape(-1, 8) # Convert each 8-bit binary value to decimal decimal_matrix = np.zeros((len(flattened_array),), dtype=int) for i, binary_value in enumerate(flattened_array): binary_string = ''.join(map(str, binary_value)) decimal_matrix[i] = int(binary_string, 2)

Reshape the result back to the original 3D shape decimal_matrix = decimal_matrix.reshape(matrix.shape[:-1]) return decimal_matrix #Decryption part def rna_computing_reverse(encoded_array, p11):

P10 = np.zeros_like(p11, dtype=np.object_)

P10[0] = encoded_array[0] u = P10.shape[0] for j in range(0, u):

Vectorized XOR operation using the truth table

```
    P10[j] = np.vectorize(lambda x, y: xor_truth_table[x][y])(encoded_array[j],
p11[j])     return P10 def reverse_rna_translation(p11):     reverse_translation_rules
= {
        'U': 'A',
        'A': 'U',
        'G': 'C',
        'C': 'G'
    }
    # Use vectorized operations for efficient reverse translation reverse_translation_func =
      np.vectorize(lambda base: reverse_translation_rules[base])     p10 =
reverse_translation_func(p11)     return p10 def reverse_mutation_rules():
    return {
        0: {'A': 'A', 'U': 'U', 'G': 'G', 'C': 'C'},
        1: {'A': 'U', 'U': 'A', 'G': 'C', 'C': 'G'},
        2: {'A': 'G', 'U': 'C', 'G': 'A', 'C': 'U'},
        3: {'A': 'C', 'U': 'G', 'G': 'U', 'C': 'A'}
    }
def reverse_rna_mutation(p10, Y):
u, v, channels, w = p10.shape
    # Reshape Y to match the shape of p10
    Y_reshaped = Y.reshape((u, v, channels, w))
```

```
    # Convert Y2 to integer Y1

    Y1 = np.floor(np.mod(Y_reshaped * 10**5, 4)).astype(int)

reverse_mutation_rules_dict = reverse_mutation_rules()      #

Use vectorized operations for efficient reverse mutation

    reverse_mutation_func = np.vectorize(lambda base, mode:
reverse_mutation_rules_dict[mode].get(base, base))

    p9  =  reverse_mutation_func(p10,  Y1)

return                              p9

reverse_transcription_table_rna_to_dna = {

    'U': 'A',

    'A': 'T',

    'G': 'C',

    'C': 'G'

}

def reverse_rna_transcription(p9):

    # Apply the reverse transcription rules using vectorization      p8 =

np.vectorize(reverse_transcription_table_rna_to_dna.get)(p9)

return    p8    def    dna_decoding(encoded_dna,    rule_index):

decoding_rules = {

        1: {'A': '00', 'T': '11', 'C': '01', 'G': '10'},

        2: {'A': '00', 'T': '11', 'C': '10', 'G': '01'},
```

3: {'A': '01', 'T': '10', 'C': '00', 'G': '11'},

4: {'A': '01', 'T': '10', 'C': '11', 'G': '00'},

5: {'A': '10', 'T': '01', 'C': '00', 'G': '11'},

6: {'A': '10', 'T': '01', 'C': '11', 'G': '00'},

7: {'A': '11', 'T': '00', 'C': '01', 'G': '10'},

8: {'A': '11', 'T': '00', 'C': '10', 'G': '01'},

}

u, v, channels, w_half = encoded_dna.shape decoded_dna =

np.empty((u, v , channels, w_half * 2), dtype=int) for i in

range(u): for j in range(v): for c in range(channels):

for k in range(w_half):

pair = encoded_dna[i, j, c, k] values =

decoding_rules[rule_index][pair] decoded_dna[i, j, c, 2 * k:2 *

k + 2] = [int(bit) for bit in values] return decoded_dna #Code for

encryption def encryption_function(image):

image_hash, hash_blocks = sha_256(image)

h1, h2, h3, h4, h5, h6 = calculate_intermediate_params(external_keys,

hash_blocks) x0,y0,z0,p0,q0=calculate_initial_values(h1,h2,h3,h4,h5,h6)

a,b,c,d=calculate_chaotic_system_parameters(h1,h2,h3,h4,h5,h6) rgb_array =

np.array(image)

Create bit planes for each channel bit_planes

= np.unpackbits(rgb_array, axis=-1)

Reshape the bit planes to form a 4D array with dimensions (height, width, channels=3, 8) bit_planes_4d = bit_planes.reshape(rgb_array.shape + (8,)) u=bit_planes_4d.shape[0] * bit_planes_4d.shape[1] *3 scrambled_bit_planes_4d = 1 - bit_planes_4d encoded_matrix = dna_encoding(scrambled_bit_planes_4d) dna_transcription_matrix = dna_transcription(encoded_matrix) Y= generate_sequences_Y(p0,q0,d,u) rna_mutation_matrix = rna_mutation(dna_transcription_matrix, np.array(Y)) rna_translation_matrix = rna_translation(rna_mutation_matrix) Z= generate_sequences_Z(p0,q0,d,u) encoded_array = rna_encoding(np.array(Z),rna_translation_matrix,u) rna_computing_matrix = rna_computing(encoded_array,rna_translation_matrix) decoding_rules = {

'A': '00',

'U': '11',

'C': '01',

'G': '10'

}

Initialize output 4D array for RGB

encrypted_matrix_shape = (rna_computing_matrix.shape[0], rna_computing_matrix.shape[1], rna_computing_matrix.shape[2], 8)

encrypted_matrix = np.zeros(encrypted_matrix_shape, dtype=int)

Map each base to binary for each channel for i in range(rna_computing_matrix.shape[0]): for j in range(rna_computing_matrix.shape[1]): for c in range(rna_computing_matrix.shape[2]): for k, base in enumerate(rna_computing_matrix[i, j, c]):

encrypted_matrix[i, j, c, k*2:k*2+2] = [int(x) for x in decoding_rules[base]] final_encrypted_matrix = binary_matrix_to_decimal(encrypted_matrix) return encrypted_matrix,final_encrypted_matrix,rna_translation_matrix def decryption_function(image,encrypted_matrix,rna_translation_matrix):

image_hash, hash_blocks = sha_256(image)

h1, h2, h3, h4, h5, h6 = calculate_intermediate_params(external_keys, hash_blocks) x0,y0,z0,p0,q0=calculate_initial_values(h1,h2,h3,h4,h5,h6) a,b,c,d=calculate_chaotic_system_parameters(h1,h2,h3,h4,h5,h6) rgb_array = np.array(image)

Create bit planes for each channel bit_planes = np.unpackbits(rgb_array, axis=-1)

Reshape the bit planes to form a 4D array with dimensions (height, width, channels=3, 8)   bit_planes_4d = bit_planes.reshape(rgb_array.shape + (8,)) u=bit_planes_4d.shape[0] * bit_planes_4d.shape[1] *3

X1, X2, X3= generate_sequences_3d_sine_chaos(x0, y0, z0, a, b, c, u)

Y= generate_sequences_Y(p0,q0,d,u)

Z= generate_sequences_Z(p0,q0,d,u)

encoding_rules = {

'00': 'A',

'11': 'U',

'01': 'C',

'10': 'G'

}

Initialize a 3D array to store the decoded RNA sequence encoded_rna_matrix=np.empty((encrypted_matrix.shape[0],encrypted_matrix.shape[1],encrypted_matrix.shape[2],encrypted_matrix.shape[3]//2), dtype='U1')

Iterate through the 3D array   for i in

range(encrypted_matrix.shape[0]):   for j in

range(encrypted_matrix.shape[1]):   for

c in range(encrypted_matrix.shape[2]):

Pair 2-bits and apply decoding rules

```
bit_pairs = ''.join(map(str, encrypted_matrix[i,

j, c, :]))

            encoded_rna_matrix[i, j, c, :] = [encoding_rules[bit_pairs[k:k+2]] for k in

range(0, len(bit_pairs), 2)]

encoded_array = rna_encoding(np.array(Z),rna_translation_matrix,u)

  decoded_rna_computing_matrix =
rna_computing_reverse(encoded_array,encoded_rna_matrix)

  reversed_rna_translated_matrix =
reverse_rna_translation(decoded_rna_computing_matrix)

  reverse_rna_mutation_matrix =
reverse_rna_mutation(reversed_rna_translated_matrix,np.array(Y))

  reverse_transcription_matrix =
reverse_rna_transcription(reverse_rna_mutation_matrix)     rule_index=1

decoded_dna_matrix = dna_decoding(reverse_transcription_matrix,rule_index)

rescrambled_bit_planes_4d = 1 - decoded_dna_matrix

  final_decrypted_matrix =
binary_matrix_to_decimal(rescrambled_bit_planes_4d)

return final_decrypted_matrix if __name__ ==

"__main__":

   main()
```

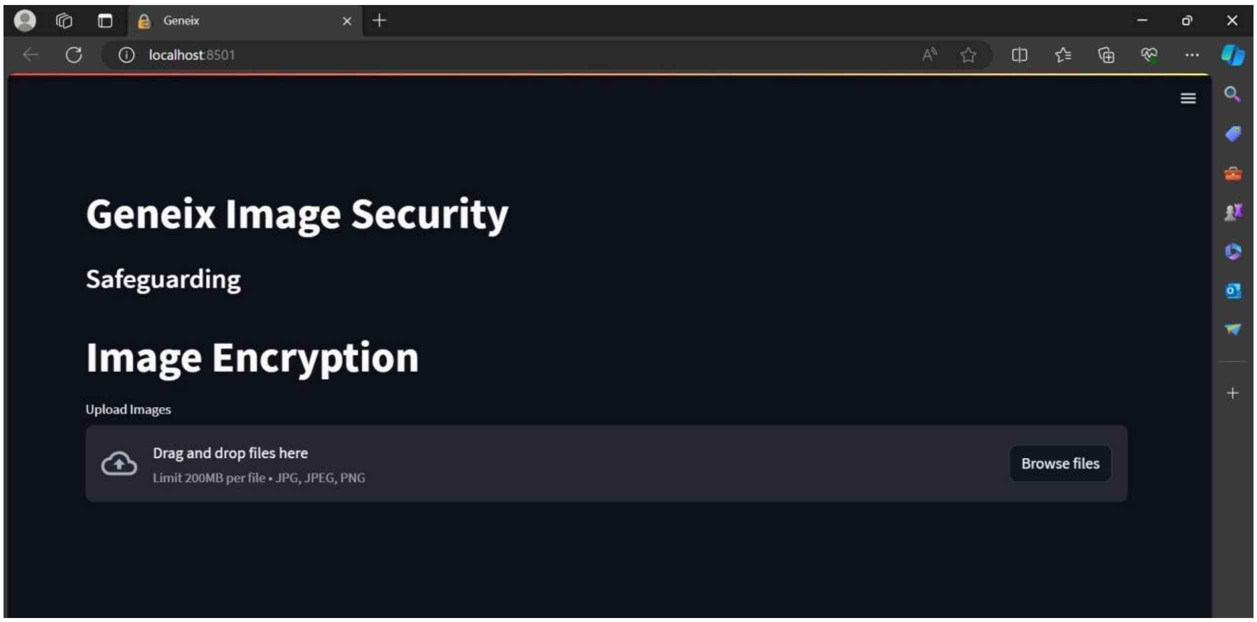
CHAPTER – 6 TESTING

6.1 TEST CASES

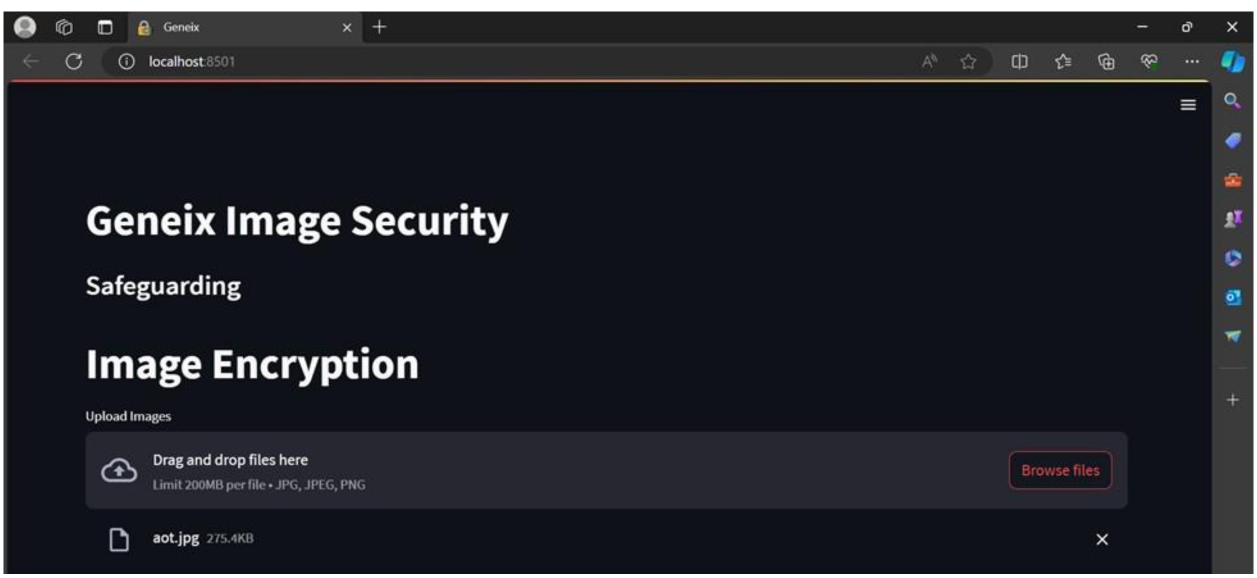
Test Case Description	Expected Outcome	Status
Upload image with valid format and dimensions	Image is successfully uploaded	Passed
Upload image with invalid format or dimensions	System displays error message	Passed
Encrypt image with valid input	Image is encrypted successfully	Passed
Encrypt image without generating key	System prompts user to generate key	Passed
Encrypt image without generating chaotic sequences	System prompts user to generate chaotic sequences	Passed
Encrypt image without applying central dogma	System prompts user to apply central dogma	Passed
Decrypt image with valid input	Image is decrypted successfully	Passed
Decrypt image without performing decryption	System prompts user to perform decryption	Passed
View encrypted image after encryption	Encrypted image is displayed to the user	Passed
View decrypted image after decryption	Decrypted image is displayed to the user	Passed

CHAPTER - 7 SCREENSHOTS

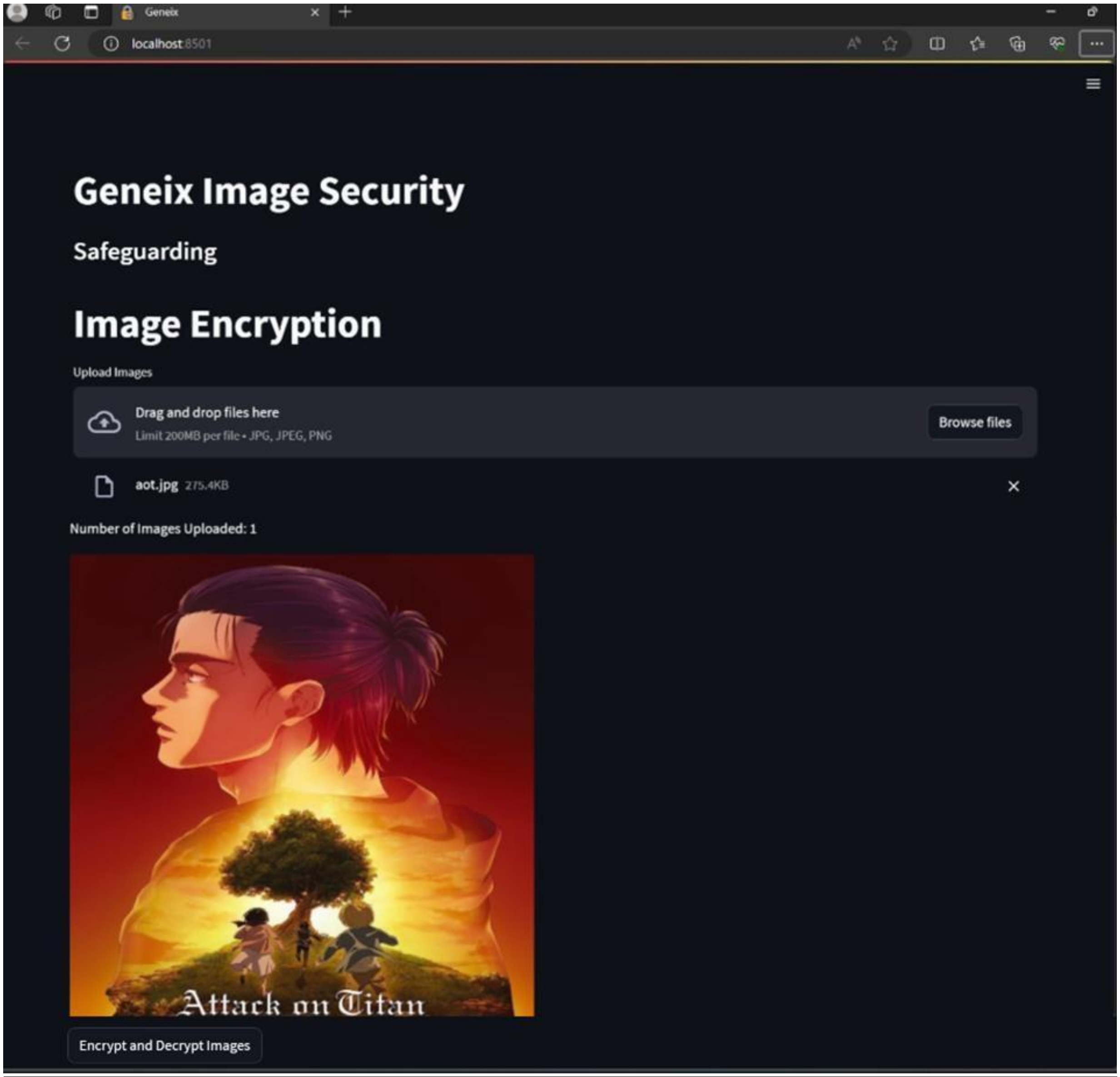
UPLOAD BOX:



IMAGES UPLOADED:

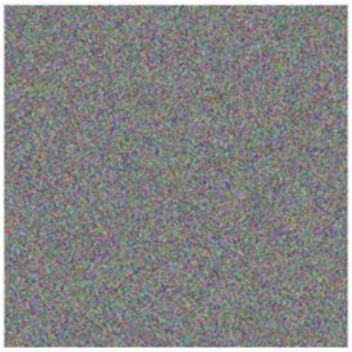


Full Display of UI:

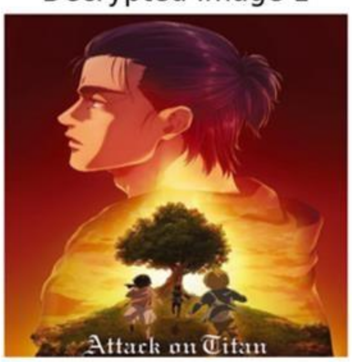


Result of Encrypted and Decrypted Images:

Encrypted Image 1



Decrypted Image 1



CHAPTER - 8

CONCLUSION AND FUTURE SCOPE

In conclusion, the fusion of chaotic systems with the central dogma presents a compelling direction for advancing security measures, especially in the context of image encryption. This innovative amalgamation offers a promising avenue for bolstering data protection by introducing unpredictability and complexity into the encryption process. By harnessing chaotic dynamics alongside the structured flow of genetic information, this approach not only enhances security but also holds potential for improving the overall efficiency and efficacy of encryption algorithms.

Moreover, the integration of biological concepts, exemplified by the central dogma, into encryption methodologies signifies a convergence of disciplines with vast implications. Beyond the realm of digital communication, this interdisciplinary synergy has the potential to catalyze transformative innovations across various domains. By drawing inspiration from nature's intricate mechanisms, we can explore novel avenues for encryption and beyond, fostering interdisciplinary collaboration and sparking creativity in fields traditionally disparate from cryptography.

Looking ahead, the adoption of this innovative paradigm opens doors to exciting future research and development opportunities. By pushing the frontiers of encryption technology, we can fortify our digital infrastructure against emerging cyber threats while also unlocking new possibilities for interdisciplinary exploration and innovation.

BIBLIOGRAPHY

[1] Central Dogma: https://byjus.com/biology/central-dogma-inheritance-mechanism/

[2] Chaos Theory: https://www.britannica.com/science/chaos-theory

[3] Chaos-based image encryption algorithm: https://doi.org/10.1016/j.physleta.2005.08.006

[4] A novel chaos-based image encryption algorithm using DNA sequence operations: https://doi.org/10.1016/j.optlaseng.2016.08.009

[5] Saberi Kamarposhti , M., Mohammad, D., Rahim, M.S.M., Yaghobi, M.: Using 3-cell chaotic map for image encryption based on biological operations. Nonlinear Dyn. 75(3), 407–416 (2014)

[6] Topics: https://atdbio.com/nucleic-acids-book/Transcription-Translation-andReplication

[7] An image encryption algorithm using two-dimensional chaotic map and DNA coding: https://ieeexplore.ieee.org/document/9687885

APPENDIX A: TOOLS AND TECHNOLOGIES

STREAMLIT: Streamlit, an open-source Python framework, streamlines the development of data-driven web applications with its intuitive interface and seamless Python library integration. Its user-friendly design empowers developers to swiftly create interactive dashboards and visualizations, facilitating efficient data exploration.

NumPy: NumPy is a powerful Python library used for numerical computing and data analysis tasks. It provides support for multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays efficiently. NumPy's array manipulation capabilities enable complex mathematical operations to be performed with ease and speed. It is widely used in various domains such as scientific computing, machine learning, image processing, and more, due to its performance benefits and ease of use.

Deployment and Version Control:

1. Git: Used for version control, allowing multiple developers to work on the project simultaneously and keep track of changes.

2. GitHub: A web-based hosting service for version control using Git, used to store the project's codebase and manage contributions.