IMAGE ENCRYPTION USING CHAOTIC LOGISTIC MAP

Project work done by

Name: Sandeep K

Reg.No: 20106108

Under the Guidance of

Ms. N.Kiruthika, MCA., M.Phil.,

Assistant Professor,

PG & Research Department of Computer Science

PG AND RESEARCH DEPARTMENT OF COMPUTER SCIENCE

SRI RAMAKRISHNA COLLEGE OF ARTS & SCIENCE (AUTONOMOUS)

Formerly SNR Sons College

Ranked 90th by NIRF 2022

Reaccredited with 'A+' Grade by NAAC

An ISO 9001: 2015 Certified Institution

AFFILIATED TO BHARATHIAR UNIVERSITY COIMBATORE – 641 006.

APRIL – 2023

CERTIFICATE

This is to certify that the project work IMAGE ENCRYPTION USING CHAOTIC LOGISTIC MAP is a Bonafide record work done by

Sandeep K

20106108

In partial fulfillment for the award of the degree of **BACHELOR OF COMPUTER SCIENCE** of Bharathiar University during January 2023 to April 2023.

Head of the Department,

Dr.G.Maria Priscilla, M.Sc., M.Phil., Ph.D.,
Associate Professor & Head,
PG & Research Department of
Computer Science
Sri Ramakrishna College of Arts & Science

Faculty Guide,

Ms.N.Kiruthika, MCA., M.Phil.,
Assistant Professor,
PG & Research Department of
Computer Science
Sri Ramakrishna College of Arts
& Science

Submitted for the Viva– Voce Examination held on _____

EXTERNAL EXAMINER **INTERNAL EXAMINER**

DECLARATION

I hereby declare that this project work entitled **"IMAGE ENCRYPTION AND DECRYPTION USING CHAOTIC LOGISTIC MAP"** submitted to Sri Ramakrishna College of Arts & Science, An Autonomous Institution, Affiliated to Bharathiar University, Coimbatore, is a record of original work done by me under the guidance of **Ms.N.KIRUTHIKA, MCA.,M.Phil** and that this project work has not formed the basis for the award of any Degree / Diploma / Associateship / Fellowship or similar to any candidate in any university.

Place: Coimbatore

Date: Signature of the Student

Countersigned by

Ms. N. Kiruthika, MCA, M.Phil.

Department of Computer Science

ACKNOWLEDGEMENT

I would like to extend my heartiest thanks to **Thiru.D.Lakshminarayanaswamy , Managing Trustee, SNR Sons Charitable Trust** with a deep sense of gratitude and respect for providing me with an opportunity to study at this institution.

I wish to record my deep sense of gratitude to our beloved **Dr.B.L.Shivakumar M.Sc., M.Phil., Ph.D., Principal & Secretary**, Sri Ramakrishna College of Arts and Science, Coimbatore for his inspiration, which make me complete this project successfully.

I take this opportunity to express our indebtedness and profound sense of gratitude to **Dr.G.Maria Priscilla, M.Sc., M.Phil., Ph.D.**, Head and Associate Professor, Department of Computer Science for her valuable help and encouragement for the successful completion of this report.

I would like to thank my professor **Ms.N.Kiruthika, MCA., M.Phil.,** who gave mean opportunity to undertake such a great challenging and innovative work. I extend my gratitude to them for their guidance, encouragement, understanding and objective criticism in the development process.

I would like to thank my Class **Tutor Dr.P.Kavitha, Ph.D.,** for her continuous support throughout my course period and for motivating me to think bigger and be unique to everyone.

Finally, I would like mention here that I am greatly indebted to each and everybody who has been associated with my project at any stage but whose name does not find a place in this acknowledgement.

ABSTRACT

An amazing digital multimedia revolution was marked at the end of the 20th century. In this revolution, images are extremely important to communicate in the present era of multimedia. In this world of hasty evolution of exchanging digital data as well as images, data protection is essential to protect data from the unauthorized parities.

With the broad use of digital images of various fields, it is important to preserve the confidentiality of image's data from any party without authorization access. Though many encryption standards and algorithms are present, it requires large computational time and high computational power for large images.

Chaotic Logistic map involves to generate a sequence of pseudo random numbers, which are then used as keys to encrypt the Image. The resulting cipher-image appears random and is extremely difficult to decrypt without the correct key. An advantage of chaotic map encryption is that it can provide a high level of security while requiring relatively simple computational resources.

TABLE OF CONTENTS

CHAPTER I

INTRODUCTION

With the rapid development of information transmission technology and popularization of multimedia capture device, multimedia data are transmitted over all kinds of wired/wireless networks more and more frequently. Consequently, the security of multimedia data becomes a serious concern of many people.

However, the traditional text encryption schemes cannot protect multimedia data efficiently, mainly due to the big differences between textual and multimedia data, such as strong redundancy between neighbouring elements of uncompressed multimedia data, bulky size of multimedia data, and some special requirements of the whole multimedia processing system.

This challenge stirs the design of special multimedia encryption schemes to become a hot research topic in multimedia processing area in the past two decades. Because there are some subtle similarities between chaos and cryptography, chaos were used to design potential secure and efficient schemes in all kinds of applications in cryptography.

According to the record of Web of Science, about two thousands of articles on chaos-based cryptology were published in the past two decades. Unfortunately, many of them have been found to have security problems of different degree from the viewpoint of modern cryptology.

In general, cryptanalysis (breaking or reporting some security defects) of a given encryption scheme requires rigorous theoretical support, so only about one tenth of the proposed chaos-based encryption schemes were reported being cryptanalyzed till now.

At the same time, the conventional cryptanalysis skills cannot be directly used to break the chaos-based encryption schemes due to some reasons, such as difference of essential structure, and some special security defects caused by the properties of the adopted chaos System. Therefore, short of security scrutiny of the proposed chaos-based encryption schemes has become bottleneck of progress of chaos-based cryptology.

Logistic map is a discrete quadratic recurrence form of the logistic equation, a model of population growth first published by P. Verhulst in 1845. The application of logistic map in cryptology can be traced back to John von Neumann's suggestion on utilizing it as a random number generator in 1947.

Due to simple form and relatively complex dynamical properties of logistic map, it was extensively used to design encryption schemes or pseudo-random number generator. Even in Web of Science, one can find about two hundred records on application of logistic map in cryptography published between 1998 and 2013. A few papers reported some security deficiencies caused by logistic map.

In an image encryption scheme based on the logistic and standard maps was proposed, where the two maps are iterated to generate some pseudo-random number sequences (PRNS) controlling twice exclusive OR (XOR) operations. It is reported that an equivalent key of the scheme can be obtained from only one known/chosen plain-image and the corresponding cipher-image.

However, the equivalent key can only be used to decrypt other cipher-images of smaller or the same size of the known/chosen plain-image. Based on a dynamical property of logistic map on stable distribution of its chaotic states, the present paper re-evaluates the security of the scheme, and finds that the scope of all the sub-keys can be narrowed much by an efficient brute-force search.

1.1 An Overview:

The logistic map is a mathematical function that can exhibit chaotic behavior. In image encryption, the logistic map can be used to generate a pseudo-random sequence of numbers that can be used as a key to encrypt an image. The Logistic map presents a polynomial map of degree two. It is a simple and one-dimensional discrete-time non-linear system. This widely used function exhibits quadratic non linearity and is defined by following equation,

$x = r * x * (1-x)$

1.2 Objectives of the project:

- Confidentiality: The primary objective of using the logistic map in image encryption is to provide confidentiality to the image data. By generating a pseudo-random sequence of numbers, the original image data can be scrambled in a manner that makes it difficult for unauthorized users to decipher.

- Security: The use of the logistic map can also provide security to the image data. The chaotic nature of the function makes it difficult for an attacker to predict the sequence of numbers that will be generated. As a result, the encryption key generated using the logistic map can be considered secure.

- Robustness: Another objective of using the logistic map in image encryption is to ensure the robustness of the encryption algorithm. The chaotic nature of the logistic map ensures that small changes in the input can result in significant changes in the output. This makes the encryption algorithm robust against attacks that attempt to reverse-engineer the encryption key or the encrypted image data.

- Efficiency: Finally, the use of the logistic map in image encryption can also improve the efficiency of the encryption algorithm. The logistic map is a simple mathematical function that can be computed quickly, making it ideal for use in real-time applications that require fast encryption and decryption.

1.3 Organization of the project:

- **Understanding the basics of logistic map image encryption**: This would involve researching the concepts and principles behind the use of the logistic map in image encryption. It would also involve understanding the mathematical formulas and algorithms used in the encryption process.

- **Implementing the encryption algorithm:** The encryption algorithm would need to be implemented using the chosen programming language and the necessary software tools.

- **Testing the encryption algorithm:** The encrypted image would need to be tested to ensure that it is secure and that the decryption process works correctly.

- **Optimizing the encryption algorithm:** The encryption algorithm would need to be optimized to ensure that it runs efficiently and can handle large images.

1.4 Scope of the system:

The scope of the system of image encryption in logistic map is to provide a secure and efficient method for encrypting digital images. This system uses the logistic map equation to generate a chaotic sequence of numbers, which is used to scramble the pixels of the original image. This scrambling process ensures that the encrypted image appears random and unintelligible to unauthorized users, while still maintaining the integrity and quality of the original image.

The system is designed to be scalable, allowing it to be used on images of various sizes and formats. It is also flexible, allowing users to adjust the encryption parameters to achieve the desired level of security and computational complexity.

The system is intended for use in various applications that require secure image transmission, such as military communications, medical imaging, and financial transactions. It can also be used in other applications, such as digital watermarking and copyright protection, where image integrity and confidentiality are critical.

Overall, the system of image encryption in logistic map offers a powerful and versatile tool for protecting the privacy and security of digital images.

CHAPTER II

SYSTEM ANALYSIS

2.1 Existing System:

Traditional encrypting mechanisms AES and RSA exhibit some drawbacks and weakness when it comes to encryption of digital images and high computing

- Large computational time for large images
- High computing power for large images Consequently, there might be better techniques for image encryption.

2.2 Proposed System:

A few chaos based algorithms provide a good combination of speed, high security complexity, low computational overheads Moreover, certain chaos-based and other dynamical systems based algorithms have many important properties such as

- sensitive dependence on initial parameters
- pseudorandom properties
- ergodicity
- non periodicity

Chaotic systems are a simple sub-type of nonlinear dynamical systems. They contain a few interacting parts which follow simple rules, but these systems are characterized by a very sensitive dependence on their initial conditions. Despite their deterministic simplicity, over time these systems can display and divergent behavior. Advantage of chaotic map encryption is that it can provide a high level of security while requiring relatively simple computational resources.

2.3 Hardware Specification:

The hardware requirements for the image encryption and decryption using chaos maps are:

CPU: Intel Core i7 or AMD Ryzen equivalent

RAM : 16GB RAM

GPU : Minimum 2 GB Dedicated graphics card of NVIDIA or AMD GPU

Storage : SSD recommended

Please note that although you could run on devices that have lower hardware specifications than this, the loading times would make it infeasible to meaningfully run, infer and fine-tune the model.

2.4 Software Specification:

The software requirements for the image encryption and decryption using chaos maps project are:

Operating System: Windows 10 or Linux

Programming Language: Python 3.x

Libraries: NumPy, Matplotlib, Pandas, PIL, os, Tkinter.

The proposed system will be developed using Python programming language, NumPy, PIL, Tkinter, Matplotlib, Pandas libraries. The system will be compatible with Windows 10 or Linux operating systems provided the dependencies are installed.

- **numpy:** This library provides support for multi-dimensional arrays and matrices, as well as many mathematical functions to operate on them.

- **matplotlib:** This library provides a way to create visualizations in Python, including line plots, scatter plots, bar charts, and more.

- **pandas:** This library provides a way to manipulate and analyze data in tabular form, including functions for reading and writing data in a variety of formats.

- **PIL:** This module provides functions to open, manipulate, and save images in a variety of file formats.

- **os:** This module provides a way of using operating system dependent functionality like reading or writing to the file system, creating processes, managing system resources, etc.

- **tkinter:** This library provides a set of tools and widgets for creating windows, menus, buttons, text boxes, and other graphical elements.

CHAPTER III

DESIGN AND DEVELOPMENT

3.1 Design Process:

- Select a suitable chaos map: The first step is to select a suitable chaos map to use for encryption. Some popular choices include the Logistic map, the Tent map, and the Hénon map.

- Choose a secret key: A secret key is needed to ensure that only authorized parties can decrypt the encrypted image. The key should be random and complex enough to resist brute-force attacks.

- Pre-processing the image: The image is pre-processed to convert it into a form suitable for encryption. This may involve converting the image to grayscale, resizing it to a standard size, or dividing it into smaller blocks.

- Generate a chaotic sequence: A chaotic sequence is generated using the selected chaos map and the secret key. This sequence is used to shuffle the image pixels and introduce randomness into the encryption process.

- Pixel shuffling: The pixels of the image are shuffled according to the chaotic sequence. This makes it difficult for an attacker to discern the original image from the encrypted one.

- Confusion and diffusion: The shuffled pixels are then subjected to confusion and diffusion operations to further obscure the image. Confusion refers to making the relationship between the pixels and the key complex, while diffusion involves spreading the image information uniformly throughout the encrypted image.

- Encryption: The final step is to encrypt the image using the secret key. The encrypted image can then be transmitted or stored securely.

- Decryption: To decrypt the image, the recipient uses the same secret key and reverse the encryption process.

Throughout the design process, it's important to consider security and performance requirements, as well as potential attacks and vulnerabilities. Thorough testing and evaluation are also necessary to ensure the efficacy and reliability of the encryption scheme.

3.2 Input Design:

- Selecting the image: The user selects the image that they want to encrypt using a Tkinter file dialog box or drag-and-drop feature.

- Choosing the encryption parameters: The user chooses the encryption parameters such as the chaos map, encryption key, and any other relevant parameters using Tkinter input widgets such as drop-down menus or text boxes.

- Generating the chaotic sequence: The chosen chaos map and encryption key are used to generate a chaotic sequence. This sequence is used to shuffle the image pixels and introduce randomness into the encryption process.

- Applying confusion and diffusion: The shuffled pixels are then subjected to confusion and diffusion operations to further obscure the image. Confusion refers to making the relationship between the pixels and the key complex, while diffusion involves spreading the image information uniformly throughout the encrypted image.

3.3 Output Design:

The output design for image encryption using chaos maps typically involves displaying the encrypted image to the user and providing feedback on the encryption process.

- Saving the encrypted image: The encrypted image can be saved to a file or database for future use. This allows the user to retrieve the encrypted image later and decrypt it using the same encryption parameters.

- Displaying the encrypted image: The encrypted image can be displayed to the user using a Tkinter image widget. This allows the user to view the encrypted image and ensure that the encryption process was successful.

Overall, the output design for image encryption using chaos maps should be user-friendly and informative. It should provide clear feedback on the encryption process and enable the user to easily access and decrypt the encrypted image.

CHAPTER IV

TESTING AND IMPLEMENTATION

4.1 System Testing :

System testing is an important part of image encryption using chaos maps. It involves testing the entire system to ensure that it is functioning correctly and securely.

- Unit testing: The individual components of the encryption system such as the pre-processing, chaotic sequence generation, pixel shuffling, and diffusion operations can be tested individually using unit testing techniques. This ensures that each component is functioning as intended.

- Integration testing: Once the individual components have been tested, they can be integrated and tested as a whole. This ensures that the components are working together correctly and that there are no conflicts or compatibility issues.

- Security testing is important to ensure that the encryption system is secure and resistant to attacks. This may involve testing for vulnerabilities such as buffer overflows or injection attacks, and verifying that the encryption process cannot be easily broken.

- Performance testing is important to ensure that the encryption system is efficient and can handle large images or a high volume of encryption requests. This may involve testing the encryption speed and memory usage.

- User acceptance testing: User acceptance testing involves testing the encryption system with actual users to ensure that it is user-friendly and meets their needs. Feedback from users can be used to improve the system and make it more intuitive and easy to use.

Overall, system testing is important to ensure that the image encryption using chaos maps system is functioning correctly and securely, and that it meets the needs of the users.

A combination of automated and manual testing techniques can be used to ensure that the system is thoroughly tested and meets the necessary requirements.

4.2 Quality Assurance:

Quality assurance is an essential aspect of image encryption using chaos maps, as it ensures that the encryption process is secure, reliable, and meets the desired quality standards.

- Test for compliance: Develop a set of test cases to ensure that the encryption process meets the quality requirements defined in the first step. This can include testing the encryption and decryption processes, testing for security vulnerabilities, and verifying that the output image quality meets the desired standards.

- Validate inputs and outputs: Validate all inputs and outputs to ensure that they are within acceptable parameters and that they are free of errors. This can include checking the size and format of the input image, the encryption key, and the output encrypted image.

- Implement error handling: Implement error handling mechanisms to detect and report any errors that may occur during the encryption process. This can help to prevent data loss or corruption and ensure that the system is functioning reliably.

- Monitor performance: Monitor the performance of the encryption system over time to identify any issues or areas for improvement. This can include tracking the encryption speed, memory usage, and error rates.

4.3 System Implementation:

- Choose a chaos map algorithm:

  Choose a chaos map algorithm to use for the encryption process (Logistic map).

- Develop the encryption algorithm:

  Implement the chosen chaos map algorithm. Define a secret key as the initial value of the logistic map. Generate a chaotic sequence of pseudo-random numbers using the logistic map by iterating the following equation

  $$x = r*x(1-x)$$

  where r is a constant that determines the behavior of the chaotic system. Select the image. Resize the input image and convert it into gray scale. Now apply XOR operation and convert the answer into decimal. This decimal number will be used as the initial input to chaotic map.

  Apply chaotic logistic map, taking initial input from the previous step, and 'r' value to be 3.95 (most chaotic).

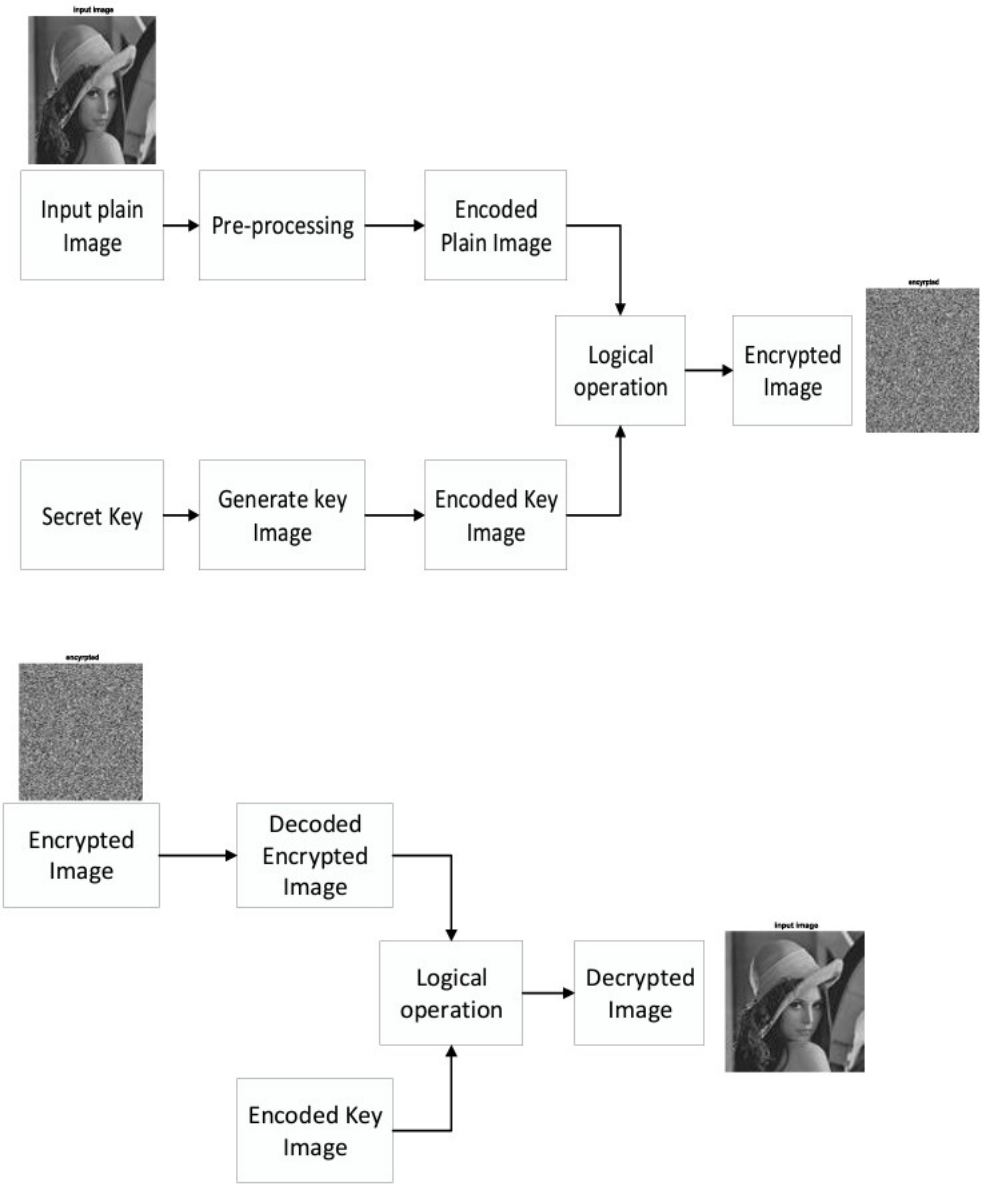
  Shuffle and substitute all the pixels in the original image with the logistic key generated.

  Save the encrypted image in a preferred location. Display the encrypted image.

  Plot the histogram of the encrypted image (if needed) for knowing the randomness of the encryption.

  Test the algorithm using sample images and verify that the encryption process is working as intended.

  Perform the decryption process by using the same generated key and XOR operation.



- Develop a user interface: Develop a user interface using Tkinter that allows users to input the image to be encrypted, select the encryption parameters such as the chaos map and encryption key, and display the encrypted image. The user interface should also have options for saving and loading encrypted images.

- Integrate the encryption algorithm with the user interface: Integrate the encryption algorithm with the user interface so that when the user selects an image and encryption parameters, the algorithm is executed and the encrypted image is displayed.

- Test the system: Test the system to ensure that it is working as intended, including unit testing of the encryption algorithm and integration testing of the user interface and encryption algorithm.

- Optimize the system: Optimize the system for performance by identifying and resolving any bottlenecks that may be affecting the encryption speed or memory usage.

- Deploy the system: Deploy the system in a production environment, such as a desktop application, depending on the intended use case.

- Maintain the system: Maintain the system by updating the encryption algorithm and user interface to address any bugs or issues that may arise and keep it up-to-date with the latest security standards.

Overall, implementing a system for image encryption using chaos maps with Tkinter as the user interface requires proficiency in Python and Tkinter programming, as well as a solid understanding of the chosen chaos map algorithm and image processing techniques.

4.4 System Maintenance:

After deploying the system for image encryption using chaos maps with Tkinter, it is important to maintain the system to ensure its proper functioning and security. Here are some maintenance tasks that should be performed:

- Keep the system up-to-date: Update the encryption algorithm and user interface to address any bugs or issues that may arise and keep it up-to-date with the latest security standards.

- Backup and restore: Perform regular backups of the encrypted images and user settings in case of a system failure or data loss. Ensure that the backup files are stored securely and can be easily restored in case of a disaster.

- Monitor system performance: Monitor the system's performance regularly to ensure that it is running smoothly and that there are no performance issues that could affect the user experience.

- Respond to user feedback: Be responsive to user feedback and respond promptly to any user complaints or issues that may arise. Use feedback to improve the user experience and address any security concerns.

- Conduct security audits: Regularly conduct security audits of the system to identify potential vulnerabilities and address them promptly.

- Provide user support: Provide user support for any questions or issues related to the system. Develop a knowledge base or help section to assist users in troubleshooting common problems.

- Plan for future updates: Plan for future updates and new features that may be added to the system. Keep up with the latest developments in encryption algorithms and image processing techniques to ensure that the system remains current and secure.

- By performing these maintenance tasks regularly, you can ensure that your system for image encryption using chaos maps with Tkinter remains reliable, secure, and up-to-date with the latest standards and best practices.

CHAPTER V

CONCLUSION

Implementing a system for image encryption using chaos maps with Tkinter as the user interface can provide a high level of security for sensitive images. The encryption process uses a chaos map algorithm to scramble the pixels of an image in a random and unpredictable way, making it difficult for unauthorized parties to decipher the original image.

The implementation process involves selecting a chaos map algorithm, developing the encryption algorithm, designing the user interface, integrating the encryption algorithm with the user interface, testing the system, optimizing its performance, deploying it in a production environment, and maintaining it over time.

By following best practices in system maintenance, such as keeping the system up-to-date, performing regular backups, monitoring system performance, and conducting security audits, you can ensure that the system remains reliable, secure, and up-to-date with the latest standards and best practices.

Overall, implementing a system for image encryption using chaos maps with Tkinter can provide a valuable security solution for images that require protection. However, it is important to carefully consider the security requirements and potential vulnerabilities of the system to ensure that it is robust and secure enough for its intended use.

5.1 Scope of Future Development:

There are several potential areas for future development in image encryption using chaos maps with Tkinter:

- Multi-layer encryption: Currently, most chaos-based image encryption systems use a single encryption layer. However, it is possible to add multiple encryption layers using different chaos maps and keys to increase the security of the system.

- Hybrid encryption: Combining chaos-based encryption with other encryption techniques, such as symmetric or asymmetric key encryption, can provide an additional layer of security and make the system more robust against attacks.

- Image compression: The ability to compress encrypted images can make the system more efficient and easier to use. Future development could focus on developing algorithms that enable efficient compression of encrypted images while maintaining their security.

- Cloud-based encryption: With the increasing use of cloud computing, there is potential for developing cloud-based image encryption systems that allow for secure storage and sharing of encrypted images.

- Machine learning-based encryption: Machine learning algorithms can be used to optimize the encryption process and increase its efficiency. Future development could explore the use of machine learning in chaos-based image encryption.

- User-friendly interface: Enhancing the user interface can improve the user experience and make the system more accessible to a wider audience.

- Real-time encryption: Real-time image encryption can be useful in applications that require continuous and secure image transmission, such as video conferencing or remote sensing. Future development could focus on developing real-time encryption algorithms that use chaos maps.

BIBLIOGRAPHY

Guo, L., & Liu, J. Image encryption algorithm based on chaotic map and Lorenz system. Journal of Physics: Conference Series, 892(1), 012096 (2017).

Li, X., Li, J., Li, W., & Chen, G.. A new image encryption algorithm based on chaotic map. Chaos, Solitons & Fractals, 36(3), 618-624 (2008).

Li, Y., Liao, X., & Xiao, D. A novel image encryption scheme based on chaotic map and Lorenz system. Optik, 125(1), 24-28 (2014).

Wang, X., Huang, W., & Tang, Z. A Novel Image Encryption Scheme Based on Logistic Map and Hyper-Chaos. Journal of Electrical Engineering and Automation, 1(1), 23-32, (2018).

References:

IEEE Xplore Digital Library (https://ieeexplore.ieee.org/)

Google Scholar (https://scholar.google.com/)

ResearchGate (https://www.researchgate.net/)

PubMed (https://pubmed.ncbi.nlm.nih.gov/)

SOURCE CODE

Logistic Key

```
def logistic_key(x, r, size):
    """
    This function accepts the initial x value,
    r value and the number of keys required for
    encryption.
    The function returns a list of pseudo-random
    numbers generated from the logistic equation.
    """

    key = []

    for i in range(size):
        x = r*x*(1-x)   # The logistic equation
        key.append(int((x*pow(10, 16))%256))    # Converting the generated number between 0
to 255

    return key
```

Logistic Map Encryption and Decryption

```
import logisticKey as key  # Importing the key generating function
from tkinter import *
import tkinter as tk
from tkinter import filedialog
import numpy as np
import os
import cv2
import matplotlib.pyplot as plt
import matplotlib.image as img
import subprocess
from PIL import ImageTk, Image


global image,image_arr,fln,pic,encryptedImage,height,width,generatedKey,enc
height = 0
width = 0




def chooseFile():
    global image,pic,fln,height,width,generatedKey,image_arr
    fln = filedialog.askopenfilename(initialdir=os.getcwd(), title='Select Image',
    filetypes=(("JPG File", "*.jpg"), ("PNG file", "*.png"), ("All Files", "*.*")))
```

```
    image = Image.open(fln).convert("RGB")
    image_arr = np.array(image)
    pic = image.load()
    print("Image Loaded")
    height, width, _ = image_arr.shape
    print(height, width)
    generatedKey = key.logistic_key(0.01, 3.95, height*width)
    print(generatedKey)



def hist():
    global generatedKey,height,width,encryptedImage
    histogram_blue = cv2.calcHist([image_arr],[0],None,[256],[0,256])
    plt.plot(histogram_blue, color='blue')
    histogram_green = cv2.calcHist([image_arr],[1],None,[256],[0,256])
    plt.plot(histogram_green, color='green')
    histogram_red = cv2.calcHist([image_arr],[2],None,[256],[0,256])
    plt.plot(histogram_red, color='red')
    plt.title('Intensity Histogram - Logistic Map Original Image', fontsize=15)
    plt.xlabel('pixel values', fontsize=10)
    plt.ylabel('pixel count', fontsize=10)
    plt.show()
# Generating dimensions of the image
```

```
# Generating keys
# Calling logistic_key and providing r value such that the keys are pseudo-random
# and generating a key for every pixel of the image

def encrypt_image():
    global image_arr,generatedKey,height,width,encryptedImage,enc
# Encryption using XOR
    z = 0

# Initializing the encrypted image
    encryptedImage = np.zeros(shape=[height, width, 3], dtype=np.uint8)

    enc = open('C:\\Users\\Sandeep\\Desktop\\Image encryptor\\enc.txt', 'w+')
    for a in generatedKey:
        enc.write(str(a) + '\n')

# Substituting all the pixels in original image with nested for
    for i in range(height):
        for j in range(width):
        # Using the XOR operation between image pixels and keys
            encryptedImage[i, j] = image_arr[i, j].astype(int) ^ generatedKey[z]
            z += 1
```

```
# Saving the encrypted image
    im = Image.fromarray(encryptedImage)
    im.save('C:\\Users\\Sandeep\\Desktop\\Image encryptor\\Logistic_Encrypted images\\encrypted_image.png')
    print("Success")
    # Displaying the encrypted image
    im.show()
def hist_enc():
    histogram_blue = cv2.calcHist([encryptedImage],[0],None,[256],[0,256])
    plt.plot(histogram_blue, color='blue')
    histogram_green = cv2.calcHist([encryptedImage],[1],None,[256],[0,256])
    plt.plot(histogram_green, color='green')
    histogram_red = cv2.calcHist([encryptedImage],[2],None,[256],[0,256])
    plt.plot(histogram_red, color='red')
    plt.title('Intensity Histogram - Logistic Map Encrypted', fontsize=20)
    plt.xlabel('pixel values', fontsize=16)
    plt.ylabel('pixel count', fontsize=16)
    plt.show()
def decrypt_image():
    global encryptedImage, generatedKey, height, width,decryptedImage
# Decryption using XOR
    z = 0
# Initializing the decrypted image
    decryptedImage = np.zeros(shape=[height, width, 3], dtype=np.uint8)
```

```
# Substituting all the pixels in encrypted image with nested for
    for i in range(height):
        for j in range(width):
            # USing the XOR operation between encrypted image pixels and keys
            decryptedImage[i, j] = encryptedImage[i, j].astype(int) ^ generatedKey[z]
            z += 1
# Displaying the decrypted image
    if decryptedImage.any():
        imf = Image.fromarray(decryptedImage)
        imf.save('C:\\Users\\Sandeep\\Desktop\\Image encryptor\\Logistic_decrypted images\\decrypted_image.png')
        print("Success")
        imf.show()
def hist_dec():
    histogram_blue = cv2.calcHist([decryptedImage],[0],None,[256],[0,256])
    plt.plot(histogram_blue, color='blue')
    histogram_green = cv2.calcHist([decryptedImage],[1],None,[256],[0,256])
    plt.plot(histogram_green, color='green')
    histogram_red = cv2.calcHist([decryptedImage],[2],None,[256],[0,256])
    plt.plot(histogram_red, color='red')
    plt.title('Intensity Histogram - Logistic Map Decrypted', fontsize=20)
    plt.xlabel('pixel values', fontsize=16)
    plt.ylabel('pixel count', fontsize=16)
    plt.show()
```

```
def open_file5():
    subprocess.Popen(["python", "C:\\Users\\Sandeep\\Desktop\\Image encryptor\\emailworking.py"])
window=Tk()
window.resizable(False, False)
window.title('Image Encryptor')
window.geometry("400x400")
canvas = Canvas(window, width=400, height=400)
canvas.grid(row=0, column=0)
# or add the canvas widget to the window using place()
canvas.place(x=0, y=0)
if canvas is None:
    print("Failed to create canvas widget")
    exit()
else:
    print("Canvas widget created successfully")
bg_image = PhotoImage(file="C:\\Users\\Sandeep\\Downloads\\uwp25419.gif")
if bg_image is None:
    print("Failed to load background image")
    exit()
canvas.create_image(0, 0, anchor=NW, image=bg_image )
tk.Button(text="Choose Image", command=chooseFile).grid(row=5, column=11, columnspan=4, padx=5,pady=50)
tk.Button(text="Histogram ", command=hist).grid(row=5, column=16, columnspan=7, padx=150,pady=50)
```

```
tk.Button(text="Encrypt", command=encrypt_image).grid(row=7, column=11,
columnspan=4, padx=10,pady=20)

tk.Button(text="Histogram", command=hist_enc).grid(row=7, column=16, columnspan=7,
padx=10,pady=10)

tk.Button(text="Decrypt", command=decrypt_image).grid(row=8, column=11,
columnspan=4, padx=10,pady=10)

tk.Button(text="Histogram", command=hist_dec).grid(row=8, column=16, columnspan=7,
padx=10,pady=10)

tk.Button(text="Send Email", command=open_file5).grid(row=9, column=15,
columnspan=4, padx=20,pady=50)

window.mainloop()
```

Email

```
import smtplib

from email.mime.multipart import MIMEMultipart

from email.mime.image import MIMEImage

import tkinter as tk

from tkinter import filedialog


class EmailSenderUI:

    def __init__(self, master):

        self.master = master

        master.title("Email Sender")
```

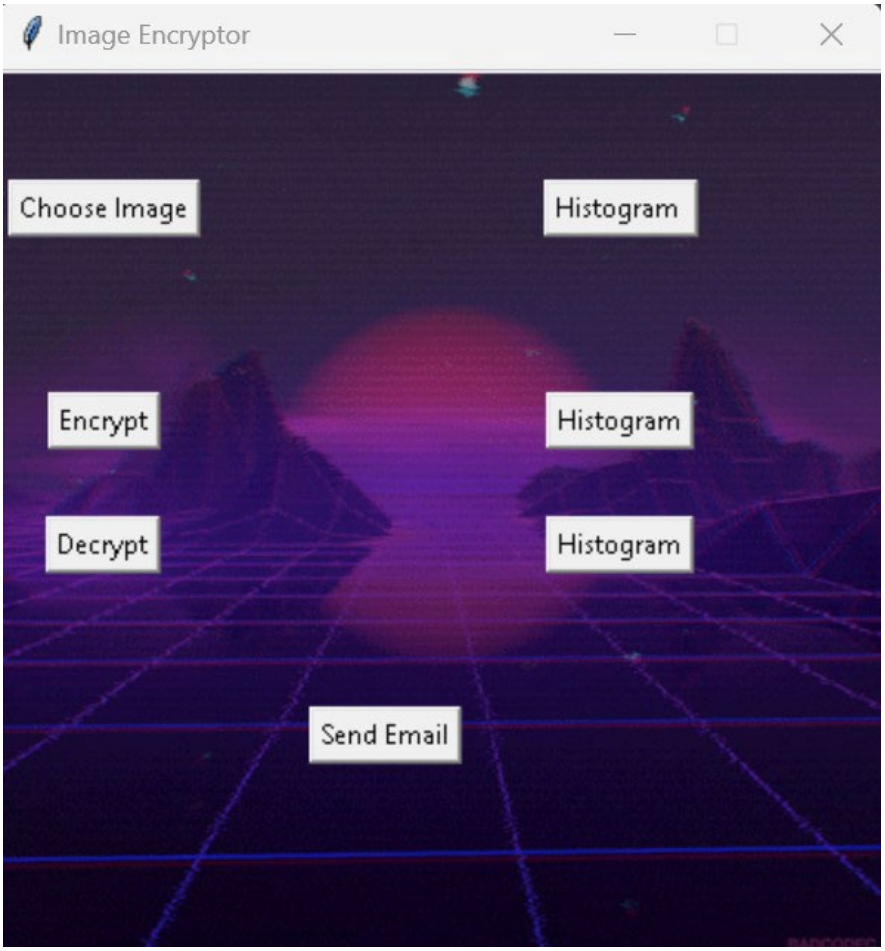
```
# email settings
self.sender_email = tk.StringVar()
self.sender_password = tk.StringVar()
self.receiver_email = tk.StringVar()
self.subject = tk.StringVar()
self.message = tk.StringVar()
self.image_path = tk.StringVar()
self.key_path = tk.StringVar()

# create widgets
tk.Label(master, text="Sender Email:").grid(row=0, column=0)
tk.Entry(master, textvariable=self.sender_email).grid(row=0, column=1)

tk.Label(master, text="Sender Password:").grid(row=1, column=0)
tk.Entry(master, textvariable=self.sender_password, show="*").grid(row=1, column=1)

tk.Label(master, text="Receiver Email:").grid(row=2, column=0)
tk.Entry(master, textvariable=self.receiver_email).grid(row=2, column=1)

tk.Label(master, text="Subject:").grid(row=3, column=0)
tk.Entry(master, textvariable=self.subject).grid(row=3, column=1)

tk.Label(master, text="Message:").grid(row=4, column=0)
tk.Entry(master, textvariable=self.message).grid(row=4, column=1)
```

```
        tk.Button(master, text="Choose Image", command=self.choose_image).grid(row=5, column=0)
        tk.Label(master, textvariable=self.image_path).grid(row=5, column=1)

        tk.Button(master, text="Choose key", command=self.choose_key).grid(row=6, column=0)
        tk.Label(master, textvariable=self.key_path).grid(row=6, column=1)

        tk.Button(master, text="Send Email", command=self.send_email).grid(row=7, column=0, columnspan=2)

    def choose_image(self):
        # open file dialog to choose image
        self.image_path.set(filedialog.askopenfilename(title="Choose Image", filetypes=[("Image files", "*.jpg *.jpeg *.png")]))

    def choose_key(self):
        # open file dialog to choose key
        self.key_path.set(filedialog.askopenfilename(title="Choose Key", filetypes=[("Key files", "*.txt")]))

    def send_email(self):
        # create message container
        msg = MIMEMultipart()
        msg['From'] = self.sender_email.get()
```

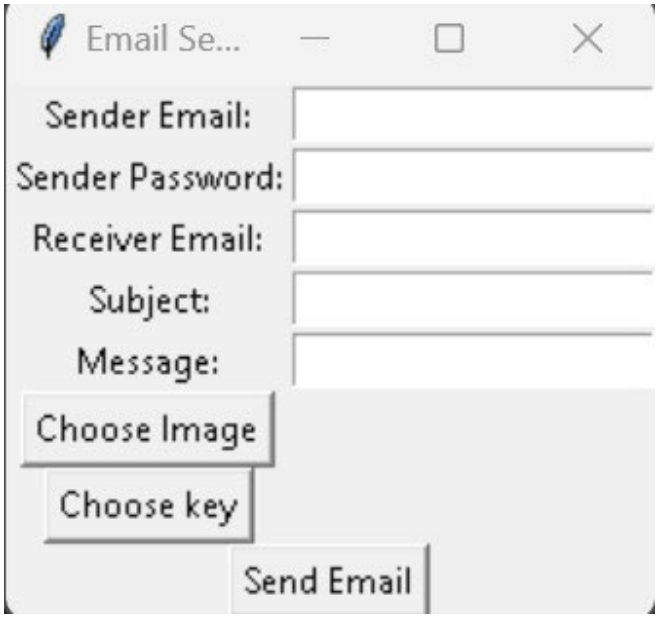
```
        msg['To'] = self.receiver_email.get()
        msg['Subject'] = self.subject.get()
        msg.attach(MIMEImage(open(self.image_path.get(), 'rb').read()))

        # create SMTP session
        with smtplib.SMTP('smtp.gmail.com', 587) as smtp:
            smtp.ehlo()
            smtp.starttls()
            smtp.login(self.sender_email.get(), self.sender_password.get())
            smtp.sendmail(self.sender_email.get(), self.receiver_email.get(), msg.as_string())

        # clear form
        self.sender_email.set('')
        self.sender_password.set('')
        self.receiver_email.set('')
        self.subject.set('')
        self.message.set('')
        self.image_path.set('')
        tk.messagebox.showinfo("Email Sent", "Email sent successfully!")

root = tk.Tk()
EmailSenderUI(root)
root.mainloop()
```

SCREENSHOTS:

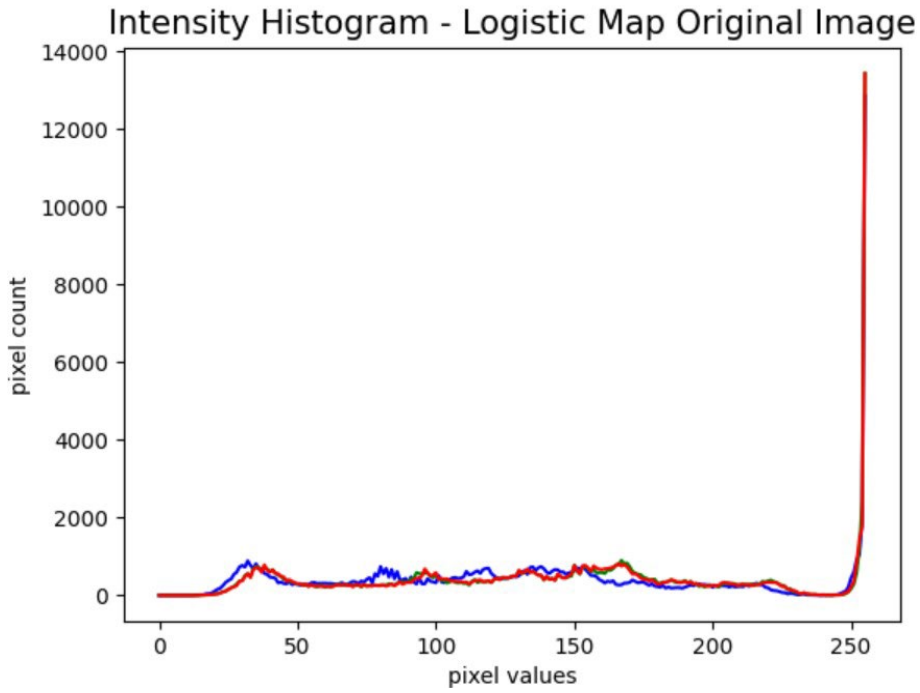
User Interface:



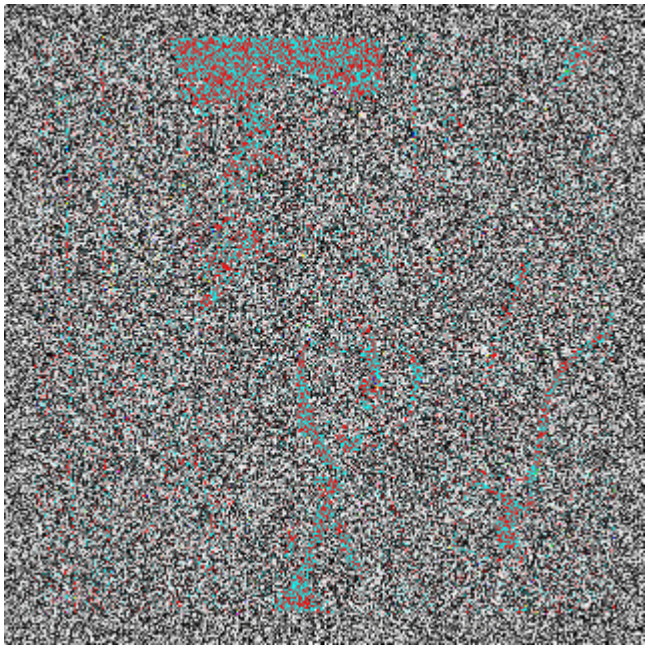
Email:

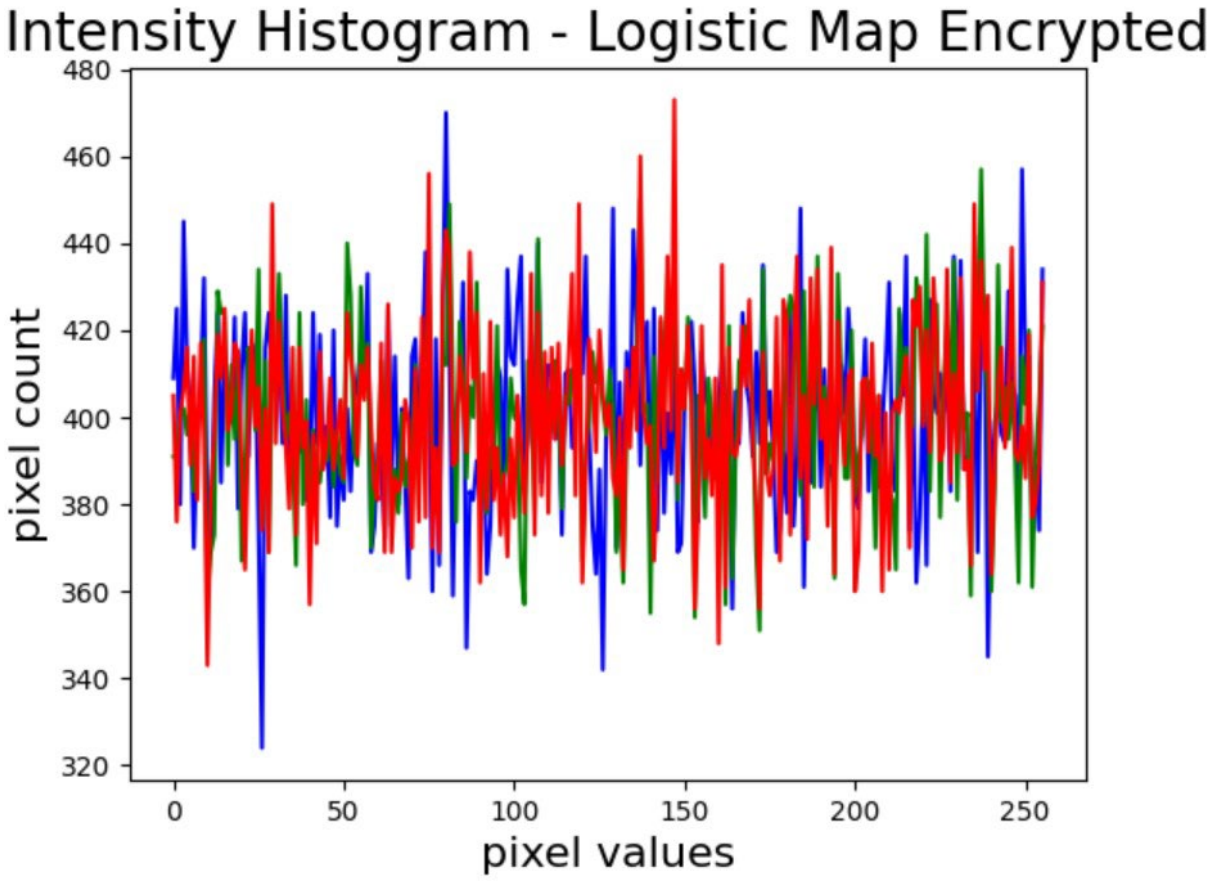


Original Image Histogram:



Encrypted Image Histogram:





Decrypted Image Histogram: