A

Project Report

On

SECRET IMAGE SHARING USING SHAMIR'S SECRET SHARING ALGORITHM

Submitted for partial fulfillment of the requirements for the award of the degree

of

BACHELOR OF ENGINEERING

In

COMPUTER SCIENCE AND ENGINEERING

By

Mr. Thimirishetty Karthikeya (2451-20-733-152)

Mr. Mohammed Rafae Ahmed (2451-20-733-162)

Under the guidance of

Mrs. Saritha Bantu
Associate Professor
Department of CSE

MATURI VENKATA SUBBA RAO ENGINEERING COLLEGE
Department of Computer Science and Engineering
(Affiliated to Osmania University & Recognized by AICTE)
Nadergul, Saroor Nagar Mandal, Hyderabad – 501 510
Academic Year: 2023-24

Maturi Venkata Subba Rao Engineering College

(Affiliated to Osmania University, Hyderabad)
Nadergul(V), Hyderabad-501510

Certificate

*This is to certify that the major-project work entitled "**SECRET IMAGE SHARING USING SHAMIR'S SECRET SHARING ALGORITHM"** is a bonafide work carried out by **Mr. Thimirishetty Karthikeya (2451-20-733-152), Mr. Mohammed Rafae Ahmed (2451-20-733-162)** in partial fulfillment of the requirements for the award of degree of **BACHELOR OF ENGINEERING IN COMPUTER SCIENCE AND ENGINEERING from Maturi Venkata Subba Rao Engineering College**, affiliated to OSMANIA UNIVERSITY, Hyderabad, under our guidance and supervision.*

The results embodied in this report have not been submitted to any other university or institute for the award of any degree or diploma.

Internal Guide	**Head of the Department**
Mrs. Saritha Bantu	Prof. J. Prasanna Kumar
Associate Professor	Professor
Department of CSE	Department of CSE
MVSREC.	MVSREC.

DECLARATION

This is to certify that the work reported in the present entitled “**SECRET IMAGE SHARING USING SHAMIR’S SECRET SHARING ALGORITHM”** is a record of bonafide work done by us in the Department of Computer Science and Engineering, Maturi Venkata Subba Rao Engineering College, Osmania University during the Academic Year 2023-24. The reports are based on the project work done entirely by us and not copied from any other source.

The results embodied in this project report have not been submitted to any other University or Institute for the award of any degree or diploma to the best of our knowledge and belief.

Thimirishetty Karthikeya	Mohammed Rafae Ahmed
(2451-20-733-152)	(2451-20-733-162)

ACKNOWLEDGEMENT

We would like to express our sincere gratitude and indebtedness to our project guide **Mrs. Saritha Bantu** for her valuable suggestions and interest throughout the course of this project.

We are also thankful to our principal **Dr. Vijaya Gunturu** and **Prof. J. Prasanna Kumar**, Professor and Head, Department of Computer Science and Engineering, Maturi Venkata Subba Rao Engineering College, Hyderabad for providing excellent infrastructure for completing this project successfully as a part of our B.E. Degree (CSE). We would like to thank our project coordinator **Mr. K Murali Krishna** for his constant monitoring, guidance and support.

We convey our heartfelt thanks to the lab staff for allowing us to use the required equipment whenever needed. We sincerely acknowledge and thank all those who gave directly or indirectly their support in completion of this work.

Thimirishetty Karthikeya (2451-20-733-152)

Mohammed Rafae Ahmed (2451-20-733-162)

VISION

- To impart technical education of the highest standards, producing competent and confident engineers with an ability to use computer science knowledge to solve societal problems.

MISSION

- To make learning process exciting, stimulating and interesting.
- To impart adequate fundamental knowledge and soft skills to students.
- To expose students to advanced computer technologies in order to excel in engineering practices by bringing out the creativity in students.
- To develop economically feasible and socially acceptable software.

PEOs:

PEO-1: Achieve recognition through demonstration of technical competence for successful execution of software projects to meet customer business objectives.

PEO-2: Practice life-long learning by pursuing professional certifications, higher education or research in the emerging areas of information processing and intelligent systems at a global level.

PEO-3: Contribute to society by understanding the impact of computing using a multidisciplinary and ethical approach.

PROGRAM OUTCOMES (POs)

At the end of the program the students (Engineering Graduates) will be able to:

1. **Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization for the solution of complex engineering problems.
2. **Problem analysis:** Identify, formulate, research literature, and analyse complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for public health and safety, and cultural, societal, and environmental considerations.

4. **Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal, and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and the need for sustainable development.
8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication:** Communicate effectively on complex engineering activities with the engineering community and with the society at large, such as being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions**.**
11. **Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Lifelong learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

PROGRAM SPECIFIC OUTCOMES (PSOs)

13. (PSO-1) Demonstrate competence to build effective solutions for computational real-world problems using software and hardware across multi-disciplinary domains.
14. (PSO-2) Adapt to current computing trends for meeting the industrial and societal needs through a holistic professional development leading to pioneering careers or entrepreneurship.

COURSE OBJECTIVES AND OUTCOMES

Course Code: PW 861 CS

Course Objectives:

- ➢To enhance practical and professional skills.
- ➢ To familiarize tools and techniques of systematic literature survey and documentation
- ➢ To expose the students to industry practices and teamwork.
- ➢ To encourage students to work with innovative and entrepreneurial ideas

Course Outcomes:

CO1: Summarize the survey of the recent advancements to infer the problem statements with applications towards society

CO2: Design a software-based solution within the scope of the project.

CO3: Implement test and deploy using contemporary technologies and tools

CO4: Demonstrate qualities necessary for working in a team.

CO5: Generate a suitable technical document for the project.

ABSTRACT

Visual cryptography is a technique that encrypts images into multiple shares, each containing partial information, such that the original image can only be reconstructed when a subset of the shares is combined. In this project, we implemented encryption and decryption algorithms for visual cryptography using Shamir's Secret Sharing Algorithm. Our proposed system addresses the flaws in existing systems by providing a straightforward, user-friendly approach to image encryption that does not require complex computations for decryption—only the correct combination of shares and the secret key is needed. This enhances the usability and practicality of the visual cryptography scheme. Through implementation and experimentation, we demonstrate the effectiveness and feasibility of the proposed visual cryptography scheme. Results show that the encrypted shares maintain confidentiality and can only be decrypted with the correct combination of shares and the secret key. This project contributes to the understanding and practical implementation of visual cryptography techniques for secure image transmission and sharing applications, building on the principles of Shamir's Secret Sharing Algorithm and extending them to visual data.

Thimirishetty Karthikeya (2451-20-733-152)

Mohammed Rafae Ahmed (2451-20-733-162)

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1
INTRODUCTION

In the era of rapidly advancing Internet technology, ensuring the security of information storage and transmission has become paramount. Cryptography stands as a cornerstone in the realm of information security, offering robust methods to protect sensitive data from unauthorized access and interception. Traditional cryptographic techniques typically involve encrypting data, rendering it unintelligible without the correct decryption key. While effective, these methods often require complex algorithms and computational resources for encryption and decryption processes.

Visual Cryptography, an emerging field in computer science, presents an innovative approach to information security. Unlike traditional cryptography, which relies on complex mathematical operations, Visual Cryptography employs a unique secret-sharing technique that encrypts images into multiple shares without the need for computational calculations during decryption. The technique was pioneered by Naor and Shamir in 1994, offering a novel way to securely transmit visual data.

At the heart of Visual Cryptography lie transparent images, each containing partial information. One image consists of random pixels, while the other holds the secret information. Individually, these images reveal no discernible content. However, when overlaid, they unveil the secret information encoded within, without the need for cryptographic keys or complex computations.

In this project, we delve into the realm of Visual Cryptography, exploring its principles and applications in securing visual data. Our focus lies on implementing a variant of Visual Cryptography that enhances security by incorporating XOR operation with a key before dividing the image into shares. By combining the principles of Visual Cryptography with XOR encryption, we aim to develop a robust method for securely encrypting and transmitting images over digital networks.

Through our implementation, we seek to showcase the practicality and efficacy of Visual Cryptography techniques in safeguarding visual data, addressing the growing concerns surrounding information security in the digital age.

1.1 Problem Statement

Traditional cryptographic methods often rely on complex algorithms and large keys to secure image data during transmission. However, these methods may face challenges in terms of computational complexity, key management, and susceptibility to cryptographic attacks. Furthermore, sharing secret keys securely among multiple parties can be cumbersome and prone to interception.

1.2 Objective

The primary objective of this project is to develop and implement a SECRET IMAGE SHARING USING SHARMI’S SECRET SHARING ALGORITH technique for secure encryption and efficient decryption of multiple images. The specific objectives include:

1. **Algorithm Implementation:** Implement encryption and decryption algorithms for visual cryptography using MATLAB, incorporating XOR operation with a key before share generation.
2. **Share Generation:** Divide the XORed image into shares using visual cryptography algorithms, ensuring data privacy and integrity.
3. **Image Reconstruction:** Develop a decryption algorithm to reconstruct the original XORed image from a subset of shares, enabling secure and efficient image transmission.
4. **Enhanced Security:** Integrate key generation and random place selection algorithms to enhance security and randomness in the encryption process, safeguarding against cryptographic attacks.
5. **Evaluation:** Evaluate the effectiveness and feasibility of the proposed visual cryptography scheme for secure image transmission and sharing, comparing it with traditional cryptographic methods.

1.3 Existing System

The existing array based visual cryptography is easy. The well-known k-out-of-n visual cryptography encoded a pixel by two arrays of sub pixels. This method encrypted a secret image among two or more sharing images which could be insignificant noise like binary images or meaningful ones. However, the process of image downscaling corrupts sharing images as well as hidden secret images seriously. For example, we employed the conventional 2-out-of-2 visual cryptography to hide a 128*128 secret image to two 256*256 sharing images. In this work, the downscaling formulation to sharing image is defined as follows: Where I (x, y) and I^(x, y) denote the pixel values at (x, y) in the original secret image and the downscaled secret image, respectively. The second row of Fig 2 shows the decrypted secret image derived from two downscaled sharing images is incorrect. It inspires us to develop Multiple Image Secret Sharing using Visual Cryptography as well as improve the capacity of hidden Images.

Drawbacks

- This system can share only 2 secret images.
- The decrypted images derived from the downscaled images are incorrect
- There is a significant loss of image quality
- It does not support proper decryption of encrypted images

1.4 Proposed System

Our proposed system leverages visual cryptography techniques to address the limitations of traditional cryptographic methods. Specifically, we implement Shamir's algorithm for visual cryptography, where the input image is XORed with a key before being divided into shares. These shares are distributed among multiple parties, and the original image can only be reconstructed when a predetermined subset of shares is combined. This approach provides visual confirmation of encrypted data and eliminates the need for sharing decryption keys among parties, thereby enhancing data privacy, integrity, and security.

1.5 Scope of Major-Project

The scope of this project encompasses the implementation of Visual Cryptography techniques for image encryption and decryption. It involves developing algorithms and software components to securely encrypt and decrypt images using MATLAB. The project's scope includes:

1. Algorithm Development: Designing and implementing algorithms for key generation, encryption, share generation, and decryption based on Visual Cryptography principles.
2. Software Development: Developing a MATLAB-based software application to automate the encryption and decryption processes. The software should provide a user-friendly interface for inputting images, specifying encryption parameters, and generating encrypted shares.
3. Security Enhancement: Exploring methods to enhance the security of the encryption process, such as incorporating additional encryption techniques or optimizing share generation algorithms.
4. Performance Optimization: Optimizing the encryption and decryption algorithms to ensure efficient processing of large image datasets while maintaining security and accuracy.
5. User Documentation: Providing comprehensive documentation and user guides to assist users in understanding the software's functionalities and operating procedures.

Input and Output:

Input:

- Original Image: The image to be encrypted and decrypted. It should be provided in a suitable format (e.g., PNG).
- Encryption Parameters: User-defined parameters such as the number of shares (n) and the threshold (k) for decryption.

Output:

- Encrypted Shares: Multiple images (share1.png, share2.png, etc.) generated during the encryption process. These shares contain partial information about the original image and are required for decryption.
- Decrypted Image: The reconstructed version of the original image obtained after decrypting the encrypted shares. It should closely resemble the original input image and reveal the hidden information encoded within the shares.

By addressing the aforementioned scope and providing clear input-output mechanisms, the project aims to deliver a robust and user-friendly solution for image encryption and decryption using Visual Cryptography techniques.

1.6 Hardware & Software Rquirments

HARDWARE:

- Minimum Requirements: The application should run on any standard personal computer with the following specifications:
- Processor: Intel Core i3 or equivalent (or AMD Ryzen 3)
- RAM: 4 GB minimum (8 GB recommended)
- Hard Drive Space: 500 MB free space
- Operating System: Windows 10 (64-bit) or macOS Monterey (or later)

SOFTWARE:

- MATLAB
- Image Processing Toolbox
- Operating System
- Text Editor or IDE

CHAPTER 2
LITERATURE SURVEY

The chapter titled "Literature Survey" serves as a comprehensive exploration of existing systems and their limitations, leading to the evolution of Visual Cryptography (VC) techniques. It begins by elucidating the fundamental concept of VC, which relies on human visual perception for decryption. The discussion then proceeds to highlight the vulnerabilities inherent in simple VC methods and proposes a solution involving embedding shares into other images using invisible digital watermarking to enhance security.

Furthermore, the chapter provides a detailed overview of the key techniques and algorithms employed in the project, including Visual Cryptography, XOR operations, randomization techniques, and Shamir's k-out-of-n VC scheme. These algorithms play a pivotal role in securely encrypting and decrypting images while ensuring simplicity and efficiency in the process.

Moreover, the chapter outlines the software technologies utilized, with MATLAB serving as the primary platform for implementation. The Image Processing Toolbox is mentioned for image manipulation tasks, along with compatibility considerations for various operating systems. Optional image editing software is also suggested for preprocessing tasks.

In essence, this chapter lays the foundation for understanding the subsequent development and implementation of the Visual Cryptography scheme for color images.

2.1 Survey of Secret Image Sharing and Security

In this section of the related work we describe the existing system, the limitation of the existing system, the earlier version, current version, proposed version, and expected results of this project Visual Cryptography is a special type of encryption technique to obscure image- based secret information which can be decrypted by Human Visual System (HVS). This cryptographic system encrypts the secret image by dividing it into n number of shares and decryption is done by superimposing a certain number of shares(k) or more. Simple visual cryptography is insecure because of the decryption process done by the human visual system. The secret information can be retrieved by anyone if the person gets at least k number of shares. Watermarking is a technique to put a signature of the owner within the creation. In this current work we have proposed a Visual Cryptographic Scheme for color images where the divided shares are enveloped in other images using invisible digital watermarking. The shares are:

Table 2.1 SURVEY ON PUBLICATIONS

S. NO	Yr. Of Pub	Author	Technique	Summary	Limitation
1	2019	Shyamalend ukandar, Arnad Maiti, Bibhas Cahndra Dhara	Visual Cryptography Scheme for Color Image Using Random Number with Enveloping by Digital Watermarking	Visual Cryptography is a special type of encryption technique to obscure image-based secret information which can be decrypted by Human Visual System (HVS). This cryptographic system encrypts the secret image by dividing it into n number of shares and decryption is done by superimposing a certain number of shares(k) or more. Simple visual cryptography is insecure because of the decryption process done by the human visual system. The secret information can be retrieved by anyone if the person gets at least k number of	It makes use of the visual cryptography technique, transform domain technique, chaos technique, noise reduction technique and error correcting code technique where the visual cryptography technique provides the capability to protect the copyright of multiple owners, and the rest of the techniques are applied to enhance the robustness of the scheme

				shares.	
2	2020	Sagar Kumar Nerella, Kamalendra Varma Gadi,Raja Sekhar Chaganti	Securing Images using Color Visual Cryptography and Wavelets	Visual Cryptography is a new Cryptography technique which is used to secure the images. In Visual Cryptography the Image is divided into parts called shares and then they are distributed to the participants. The Decryption side just stacking the share images gets the image. The initial model developed only for the bi-level or binary images or monochrome images. Later it was advanced to suit for the Colour Images means Gray Images and RGB/CMY Images. For the RGB/CMY Images different methods are developed based on the colour decomposition techniques. In this paper we propose a new way of performing colour visual cryptography	The important feature of the Visual Cryptograph y is decryption doesn't require any computer and it requires less computation al power. The Image is divided into parts called shares and then they are distributed to the participants.

				using wavelet technique. Wavelet techniqueis used to convert the Colour Image to Gray	
3	2021	Yi-Chong Zeng,and Chi-Hung Tsai	High Capacity Multiple Image Sharing Scheme by combining Visual Cryptography and Data Hiding	The proposed multi-scale image sharing scheme to hide multiple images to two meaningful shares. The proposed method combines conventional 2-out-of-2 visual cryptography with a data hiding technique. The overall effort of the proposed scheme is the achievement of decrypting or extracting multiple secret images and reference images from sharing images at different scale- levels. More Over, it is not only applied to monochromatic images but also color halftone images. The experimental results will demonstrate that the proposed scheme indeed increases more capacity of hidden images than the	The overall effort of the proposed scheme is the achievement of decrypting/e xtracting multiple secret imagesand reference images from sharing images at different scale-levels. Moreover, itis not only applied to monochromatic images but also color halftone images. The experiment results will demonstrate that the proposed scheme indeed increases more capacity of hidden images than the conventional2-out-of-2 visual cryptography does.

				conventional 2-out-of-2 visual cryptography does.	
4	2022	Bindu Rani, Mini Agarwal, Ramakanth	Improvement Hybrid of Visual Cryptograpy and RSA for Color Image Security	A digital image is a numeric representation of a two- dimensional image. Depending on whether the image resolution is fixed, it may be of vector or raster type.	The massive pixel confusion by non-linear Hénon map gives rise to an efficient.

2.2 Techniques & Algorithms of Secret Image Sharing

In this project, several techniques and algorithms are employed to implement Visual Cryptography for image encryption and decryption. Below is a brief overview of the key techniques used:

1. Visual Cryptography: Visual Cryptography is a cryptographic technique that allows the encryption of images into multiple shares, such that combining these shares reveals the original image. It relies on the visual perception of humans to decrypt the information without complex computations.[2]

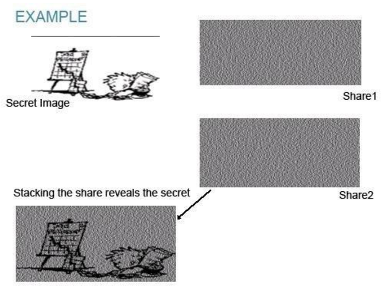
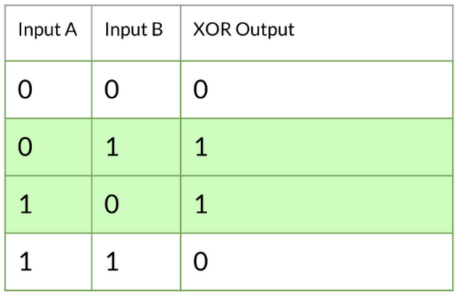


Fig 2.1 Demonstration of visual cryptography

2. XOR Operation: XOR (exclusive OR) operation is a bitwise operation used extensively in Visual Cryptography. It combines two binary inputs and produces an output where each bit is set if the corresponding bits in the input differ. XOR is

used in key generation, encryption, and decryption steps to introduce randomness and ensure security in the cryptographic process.[3]



Input A	Input B	XOR Output
0	0	0
0	1	1
1	0	1
1	1	0

Fig 2.2 XOR Operation in binary inputs

3. Randomization Techniques: Randomization techniques are employed to enhance the security of the encryption process. For example, randomization may involve shuffling the order of pixels or introducing noise into the encrypted shares to prevent unauthorized access to the original image.

Shamir's k-out-of-n Visual Cryptography: Also known as Shamir Secret Sharing (SSS) is a cryptographic technique that protects sensitive data by distributing data fragments across multiple parties.[4] This is a specific type of VC scheme developed by Adi Shamir. It allows you to create n shares from a secret image, where any k out of those n shares can be used to recover the original image. Here's a breakdown of the parameters:

- k (threshold): This value defines the minimum number of shares required to recover the secret image. It ensures a level of security as someone with fewer than k shares cannot glean any information about the original image.
- n (number of shares): This value specifies the total number of shares created from the secret image. Having more shares (while maintaining a reasonable k value) can provide additional security by distributing the information across more pieces.

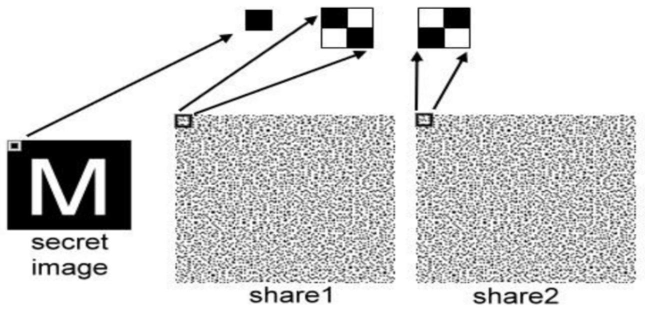


Fig 2.3 Shamir’s Visual Cryptography schema

Algorithms of Shamir’s Visual Cryptography:

The core algorithm in this project revolves around Shamir's k-out-of-n Visual Cryptography scheme. This scheme mathematically transforms the secret image into shared images using random numbers. Here's a simplified overview of the steps involved:

1. Key Generation:

   Key generation is the process of creating a secret key used for encrypting and decrypting images. In this project, a key generation algorithm is implemented to generate a key matrix with integer values based on a user-provided key string.
2. Encryption Algorithm:

   The encryption algorithm converts the original image into a binary form and performs bitwise operations (e.g., XOR) between the binary image and the key matrix. This process generates encrypted shares, ensuring that each share individually reveals no discernible information about the original image.
3. Share Generation:

   Share generation involves dividing the encrypted binary image into distinct segments, each containing partial information about the original image. The shares are designed such that combining them reveals the original image, while individual shares provide no insight into the original content.
4. Decryption Algorithm:

   The decryption algorithm reconstructs the original image from the encrypted shares. It overlays the share images and performs bitwise operations (e.g., XOR) between corresponding pixels to recover the original binary image. The decrypted binary image is then converted back to its original format to obtain the final decrypted image.

By integrating these techniques and algorithms, the project achieves robust image encryption and decryption capabilities while maintaining simplicity and efficiency in the cryptographic process.

2.3 Software Technologies

1. MATLAB:
    - MATLAB serves as the primary platform for implementing the visual cryptography algorithms and encryption/decryption processes.
    - Version: MATLAB R2017 or later.
2. Image Processing Toolbox:
    - The Image Processing Toolbox in MATLAB provides essential functions and tools for manipulating and processing images.
    - Version: Compatible with MATLAB R2017 or later.
3. Operating System:
    - The software can run on various operating systems, including Windows, macOS, and Linux.
    - Ensure compatibility with the chosen operating system for MATLAB installation.
4. Text Editor or IDE:
    - A text editor or integrated development environment (IDE) may be required for editing and managing MATLAB script files.
    - Recommended options include Notepad++ (for Windows), Sublime Text, or MATLAB's built-in editor.
5. Image Editing Software:
    - These tools can assist in adjusting image dimensions, color balance, and other parameters to ensure compatibility with the encryption process.

CHAPTER 3
SYSTEM DESIGN

The "System Design" chapter constitutes a critical phase in the project's development, focusing on structuring the Visual Cryptography (VC) system through Unified Modeling Language (UML) diagrams and defining its architecture. Beginning with an overview of the project's objectives, this chapter delves into the systematic breakdown of the VC system's components and interactions.

UML diagrams, including Use Case diagrams, Class diagrams, Sequence diagrams, and Activity diagrams, are employed to depict the system's functionalities, relationships, and workflows comprehensively. These diagrams provide a visual representation of how users interact with the system and how different modules collaborate to achieve the desired outcomes.

Furthermore, the chapter elucidates the system architecture, outlining the high-level design decisions and the distribution of responsibilities among various components. It discusses key architectural patterns, such as client-server architecture or layered architecture, chosen to ensure scalability, maintainability, and reliability of the VC system.

3.1 UML Diagrams of Shamir’s Secret Sharing Algorithm

Use Case Diagram:

The use case diagram illustrates the process of encrypting and decrypting an image, highlighting the interactions between the sender, the receiver, and the system's functionalities. The sender initiates the process by selecting the image to be encrypted and providing the necessary key for encryption. The image is then encrypted, generating multiple shares as part of the encryption process. This ensures that the image data is securely transformed. The receiver, on the other hand, is responsible for selecting the necessary shares and providing the key to decrypt the image, thus recovering the original image from the encrypted data. The diagram showcases the flow from selecting an image to encrypting it, generating shares, and ultimately decrypting it to retrieve the original image.

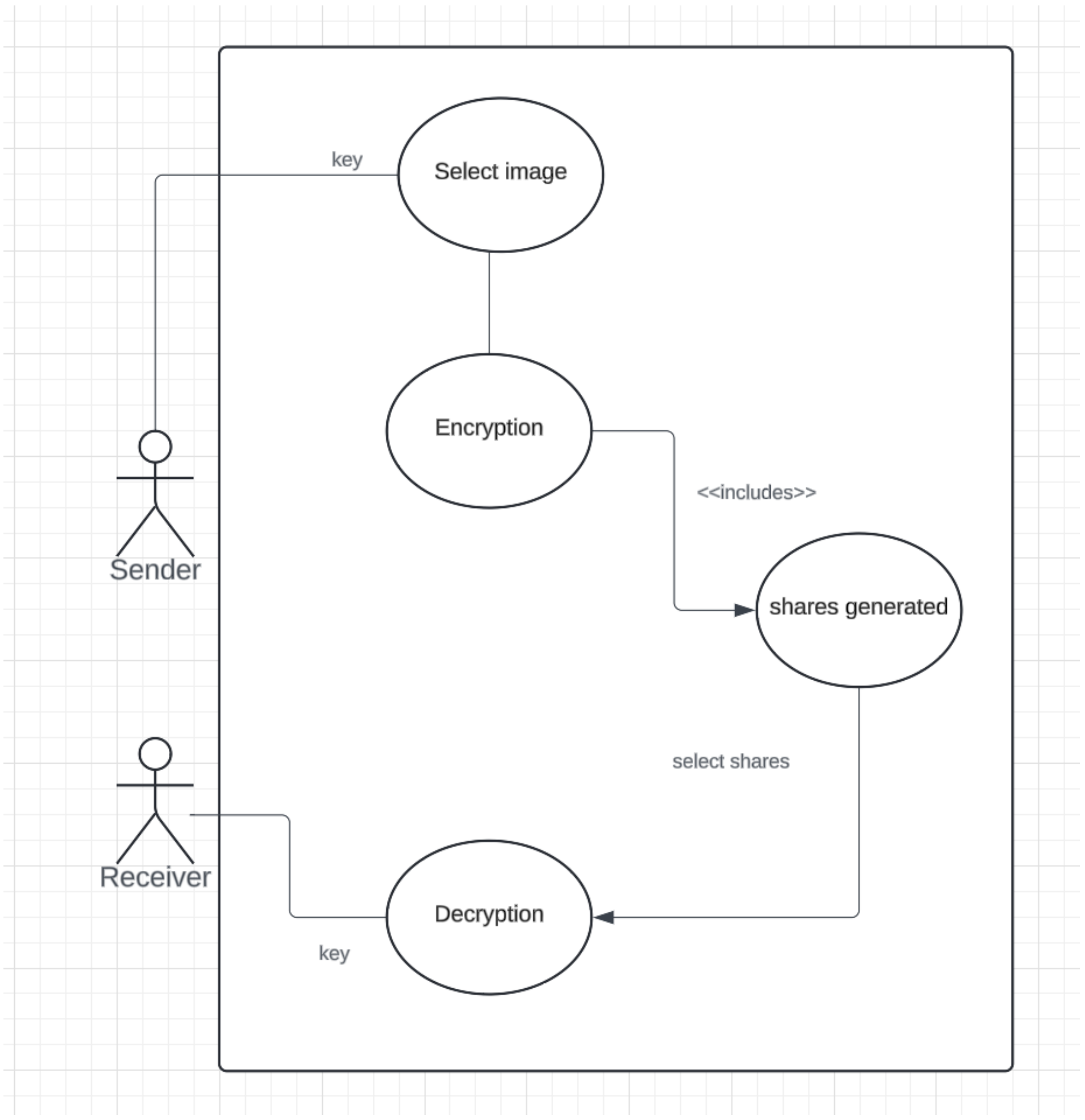


Fig 3.1 USE CASE DIAGRAM OF SECRET IMAGE SHARING

CLASS DIAGRAM:

The class diagram illustrates a visual cryptography system of the project. Users interact with this system to encrypt or decrypt images. The core functionality resides in the VisualCryptographySystem class, which utilizes EncryptionAlgorithm and DecryptionAlgorithm classes for the respective tasks. These algorithms rely on the Image class to manage the image data. This breakdown highlights the key components and their interactions within the visual cryptography system.

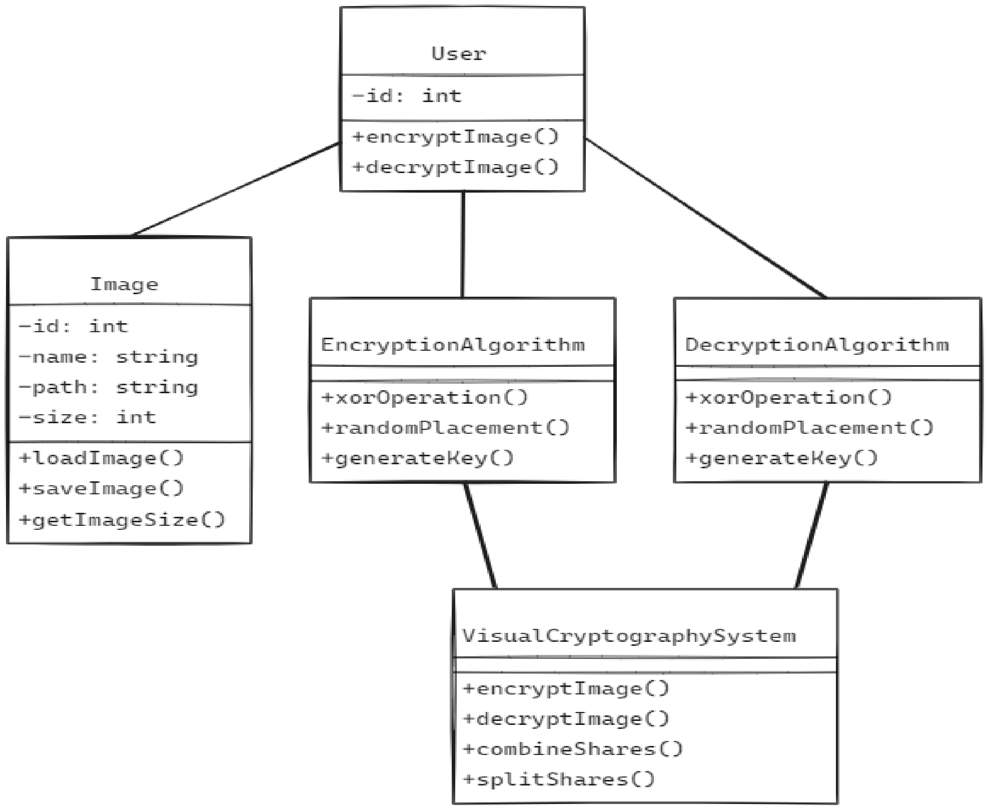


Fig 3.2 CLASS DIAGRAM OF SECRET IMAGE SHARING

STATE DIAGRAM:

The state diagram for the "Generate Shares" phase depicts the system's states and transitions during the process. It begins in the state of "InitializingShares," where the shares data structures are set up. Then, it enters the state of "LoopingThroughPixels" to process each pixel sequentially. Within this loop, it transitions to "GeneratingPositions" to generate random positions for the current pixel, followed by "DistributingPixel" to distribute the pixel value among the shares. This loop continues until all pixels are processed, transitioning back to "LoopingThroughPixels" as needed. Finally, once all pixels are processed, it moves to the state of "SavingShares" to save the generated shares as image files, marking the end of the share generation process.

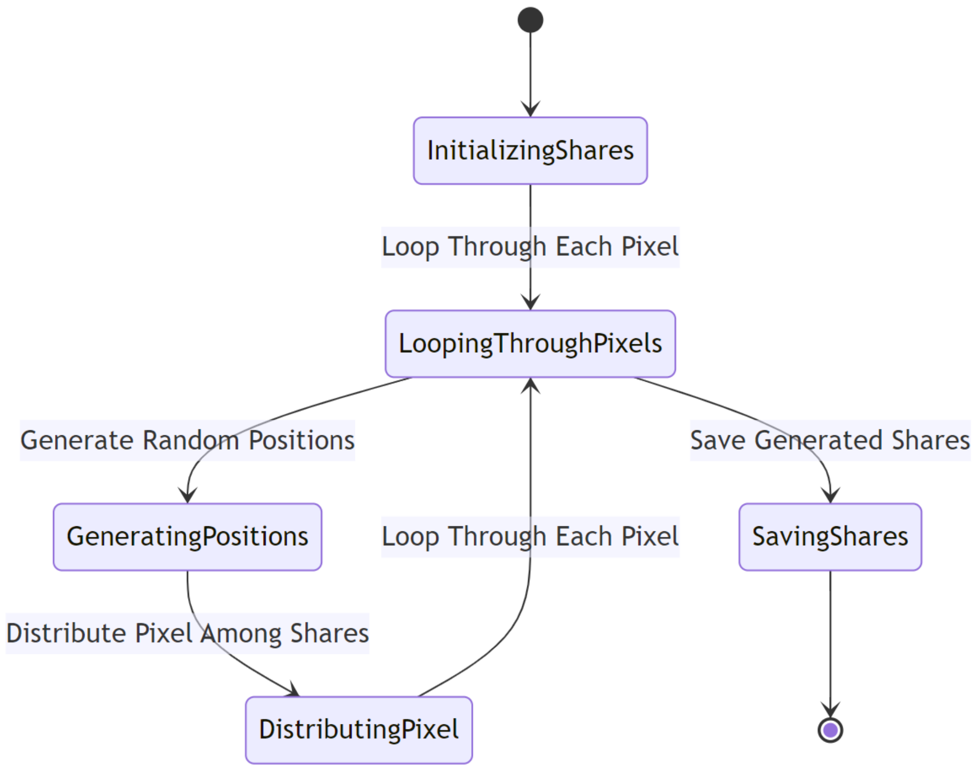


Fig 3.3 State Diagram of Share Generation phase

ACTIVITY DIAGRAM:

The activity diagram represents the "Generate Shares" phase, illustrating the process of splitting an encrypted image into multiple shares. It begins by initializing data structures for holding shares and then iterates through each pixel in the image. For each pixel, it checks the value, generates random positions for distributing the pixel value among shares, and assigns the pixel value to these positions. Once all pixels have been processed, the generated shares are saved as image files, concluding the share generation process.

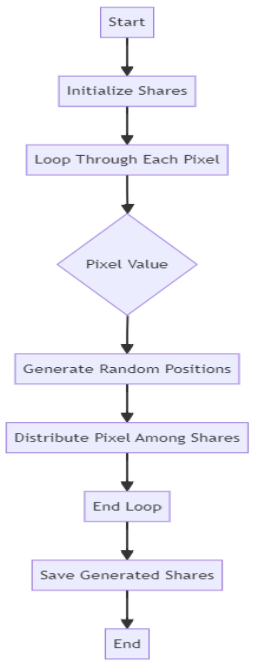


Fig 3.4 Activity Diagram for Shares Generation Phase

SEQUENCE DIAGRAM:

The sequence diagram you provided illustrates the interaction between various components in a visual cryptography system for encryption and decryption. Here's a concise explanation:

The User initiates the process, indicating whether they want to encrypt an image or decrypt shares.

Visual_Cryptography_System, the central coordinator, interacts with appropriate algorithms based on the user's choice.

During Encryption:

1. Visual_Cryptography_System calls upon the **Encryption_Algorithm**.
2. The Encryption_Algorithm generates a secret key, encrypts the image using XOR, and randomly distributes encrypted data across multiple shares.
3. Visual_Cryptography_System then saves these shares as separate images.

For Decryption:

1. Visual_Cryptography_System interacts with the **Decryption_Algorithm**.
2. The Decryption Algorithm recovers the key and performs XOR operations on the shares (assumed to be provided by the user).
3. While the diagram doesn't explicitly show it, the decrypted shares are likely presented to the user for reconstruction of the original image.

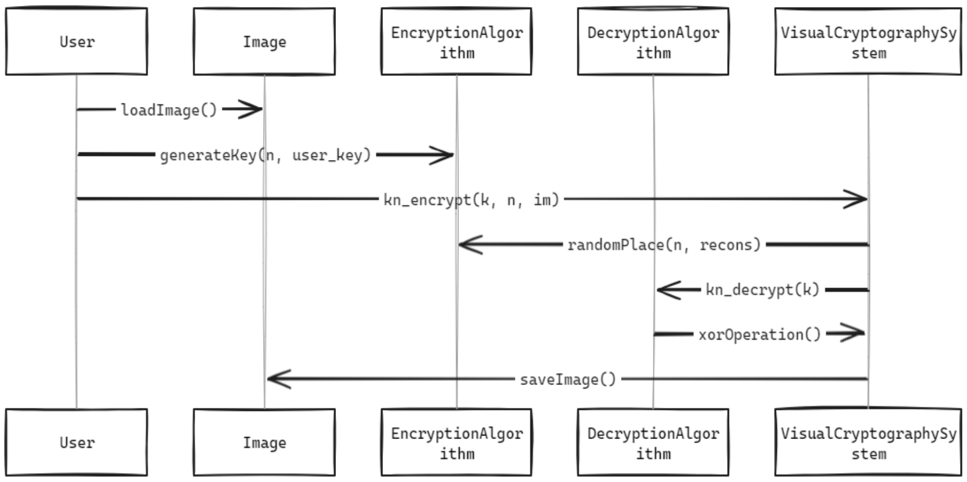


Fig 3.5 SEQUENCE DIAGRAM OF SHAMIR'S ALGORITHM

3.2 System Architecture of Shamir's Secret Sharing Algorithm

The architecture of the project involves splitting an input image into multiple shares (share1, share2, ..., shareN) using the algorithm, where a threshold number k out of n shares are required to reconstruct the original image.

The key steps are as follows:

1. An input key is generated based on user parameters like width and height.
2. The input image is converted to RGB format.
3. The RGB values are converted to binary format.

4. Shamir's Secret Sharing Algorithm is applied to split the binary data into n shares, where k shares are randomly selected for reconstruction.
5. During reconstruction, the selected k shares are combined using an XOR operation.
6. The resulting binary output is reshaped to the original dimensions, producing the output image.

This visual cryptography scheme provides a secure way to share and transmit sensitive image data by dividing it into multiple shares, where access to a sufficient number of shares is required for reconstruction. The algorithm ensures that individual shares do not reveal any information about the original image, enhancing data security and privacy.

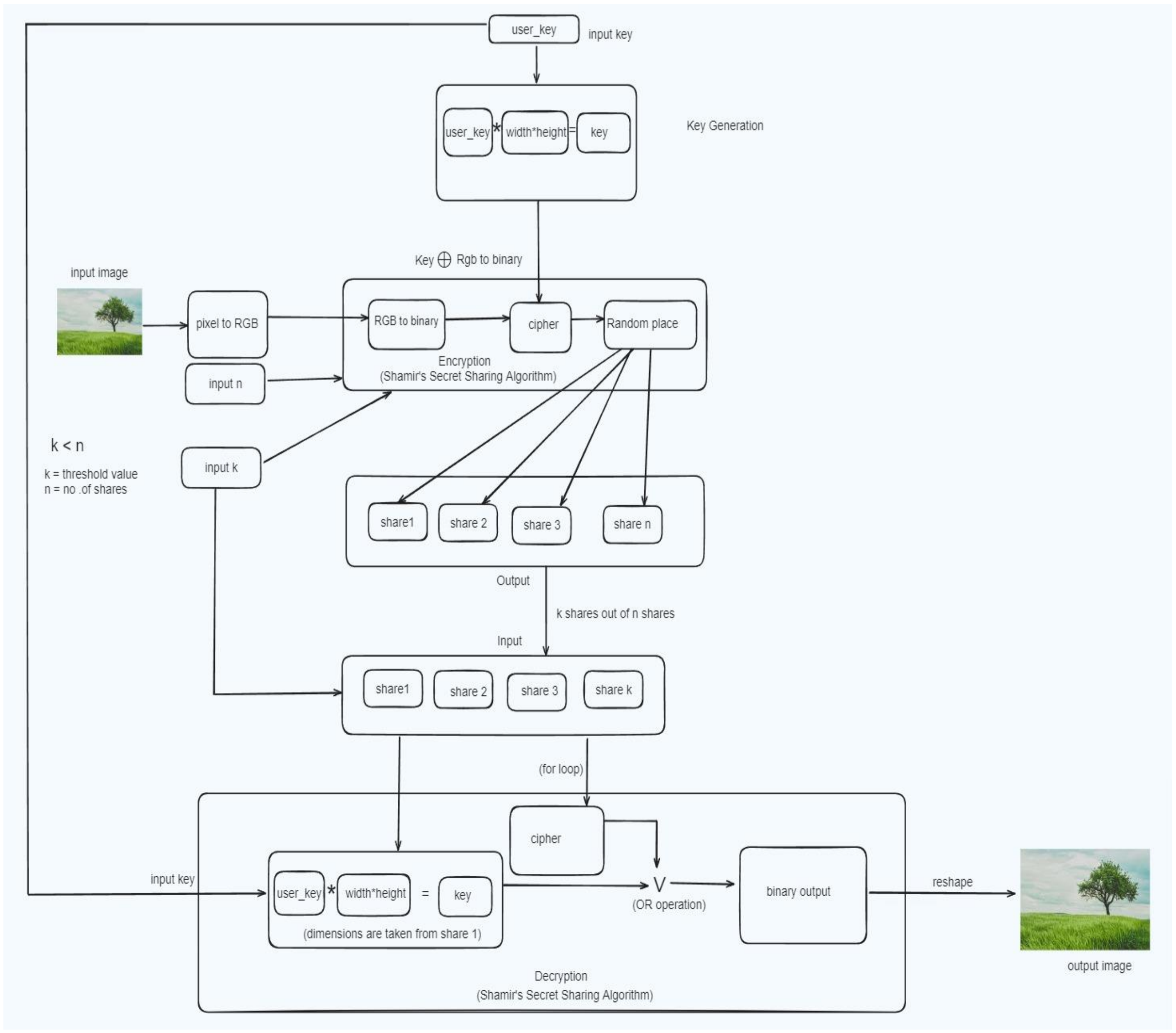


Fig 3.6 SYSTEM ARCHITECTURE OF SECRET IMAGE SHARING

3.3 Project Plan for Secret Image Sharing

The proposed project on Visual Cryptography is structured into six comprehensive phases to ensure a systematic and thorough development process. The following phases are:

Starting with Research and Scope Definition, the initial phase focuses on gathering essential knowledge about Visual Cryptography and outlining the project's objectives and requirements within a month. The subsequent Hardware and Software Tools Setup phase, lasting two weeks, involves identifying the necessary tools and configuring the development environment, including the installation of MATLAB and its Image Processing Toolbox. In the third phase, spanning four weeks, the core image encryption algorithm will be developed and rigorously tested, alongside the implementation of key generation and random placement functions. The fourth phase will focus on share generation and storage, ensuring that the encrypted shares are correctly generated and stored as image files over four weeks. Following this, the Decryption Algorithm and GUI Development phase will last four weeks, during which the decryption algorithm will be created, and an intuitive GUI will be designed for user interaction. Finally, the project will undergo a four-week period of testing, optimization, and finalization to ensure all components function accurately and efficiently, culminating in a polished and deployable Visual Cryptography solution.

Phases and Timeline:

Phase 1: Research and Scope Definition (1 month)

Tasks:

- Research on Visual Cryptography and its applications.
- Define project scope, objectives, and requirements.
- Draft project abstract and title.

Duration: 1 month

Phase 2: Hardware and Software Tools Setup (2 weeks)

Tasks:

- Identify hardware and software requirements.

- Install MATLAB and configure the Image Processing Toolbox.
- Set up development environment, including necessary software tools and libraries.

Duration: 2 weeks

Phase 3: Image Encryption Algorithm Development (4 weeks)

Tasks:

- Develop and test the image encryption algorithm.
- Implement the key generation algorithm (keyGen).
- Validate the random placement function (randomPlace).

Duration: 4 weeks

Phase 4: Share Generation and Storage (4 weeks)

Tasks:

- Implement the share generation algorithm (kn_encrypt).
- Develop functionality to save generated shares as image files.
- Ensure shares can be accurately generated and stored.

Duration: 4 weeks

Phase 5: Decryption Algorithm and GUI Development (4 weeks)

Tasks:

- Develop the decryption algorithm (kn_decrypt).
- Design and implement the GUI for decryption (decryptgui).
- Integrate the decryption process with the GUI for user interaction.

Duration: 4 weeks

Phase 6: Testing, Optimization, and Finalization (4 weeks)

Tasks:

- Perform thorough testing of both encryption and decryption processes.
- Optimize algorithms for efficiency and accuracy.
- Make necessary adjustments and finalize the project for deployment.

Duration: 4 weeks

CHAPTER 4
SHAMIR'S SECRET SHARING IMPLEMENTATION & METHODOLOGIES

The chapter "Shamir's Secret Sharing Implementation and Methodology" details the practical execution of the Visual Cryptography (VC) system, covering essential stages such as input image preparation, key generation, encryption, share generation, decryption, and output generation. Each step is crucial for the system's functionality, ensuring the security and integrity of the encryption and decryption processes. Through MATLAB implementation, the chapter provides a concise yet comprehensive guide to realizing the VC system's conceptual framework into a functional reality.

4.1 Environment Setup

To ensure the successful implementation and execution of the project, it is essential to set up a robust development environment. This project utilizes MATLAB due to its powerful computational capabilities and the comprehensive suite of tools it offers, including the Image Processing Toolbox. MATLAB provides an efficient environment for image manipulation and processing, which are critical for our encryption and decryption algorithms. Below are the step-by-step guidelines for installing and configuring the necessary software components to create an optimal development setup.

Installation Guidelines of MATLAB Software

MATLAB Installation:

- Visit the MathWorks website and log in to your account.
- Download the MATLAB installer appropriate for your operating system (Windows, macOS, or Linux).
- Run the installer and follow the on-screen instructions.
- During installation, choose the option to install all components, including the Image Processing Toolbox.
- Complete the installation process and activate MATLAB using your MathWorks account credentials.

Configuration of Image Processing Toolbox:

- Once MATLAB is installed, launch the software.
- Navigate to the "Add-Ons" tab in the MATLAB toolbar.
- Select "Get Add-Ons" to open the Add-On Explorer.
- Search for "Image Processing Toolbox" in the Add-On Explorer.
- Click on the Image Processing Toolbox and select "Install".
- Follow the prompts to complete the installation of the toolbox.

Compatibility Check:

- Before proceeding, ensure that MATLAB and the Image Processing Toolbox are compatible with your operating system.
- Verify that your operating system meets the minimum requirements for running MATLAB.
- Check for any compatibility issues between MATLAB and your operating system version.

4.2 Pseudo Code

❖ STEP1: Process the image in pixel to binary format of size same as the input image.

❖ STEP2: Fix the threshold value, shares to be generated and key. (k, n, key)
Note: k is always less than or equal to n

❖ STEP3: Generate the key to be used for encryption and decryption which is of size of processed image.

❖ STEP4: Encryption Process:
4.1: Perform XOR operation between each pixel of processed image and the corresponding of the key matrix.

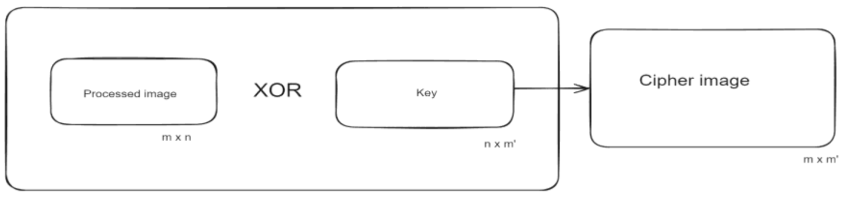


Fig.4.1 Encryption of input image

4.2: Divide encrypted binary image into distinct segments, each containing partial information.

4.3 Ensure that each shared image contains a portion of the encrypted data but reveals no information about the original image.

4.4Save each share as a separate image file (e.g., share1.png, share2.png). Shares are distributed among k number of receivers.

Note: If only one share is used to decrypt, the output is not retrieved as original input image as it was at sender side. Therefore, n is always greater than 1.

❖ STEP5: Decryption Process:

Note: In this module, above k number of shares to be used by n number of users, in order to retrieve the original image.

5.1: Reconstruct the original binary image using for loop from the combined shares.

5.2: Perform bitwise operation (OR) between corresponding pixels of shared images and key

5.3: Convert the binary image back to original format.

Advantages:

1. When using symmetric key for encryption, there is a chance of key getting stolen. So in Shamir’s Secret Sharing Algorithm, even if the key is stolen, it would be useless without the shares.
2. Complexity can be increased by increasing the number of shares.

4.2 Module Implementation Description

The implementation phase of the secret sharing using shamir's secret sharing algorithm project involves the development of encryption and decryption algorithms, key generation, and random placement techniques using MATLAB. This section details the step-by-step process of how the input image is securely divided into shares and subsequently reconstructed. The use of MATLAB's Image Processing Toolbox enhances the efficiency and effectiveness of these operations, ensuring that the visual cryptography techniques are accurately executed. Through this implementation, the project aims to demonstrate the practical application and security benefits of visual cryptography in image transmission and sharing.

1. **Input Image Preparation**:

   The system begins by preparing the input image that needs to be encrypted. Ensure that the input image is in a suitable format (e.g., PNG) and meets the required dimensions for processing. Convert the original image into binary form. (e.g., RGB to Binary)

   Note: For a color image, each pixel is represented by three values i.e., R,G,B. and it is stored in a 3D matrix. Each of these RGB values is then converted to binary form with the help of the function 'dec2bin'.

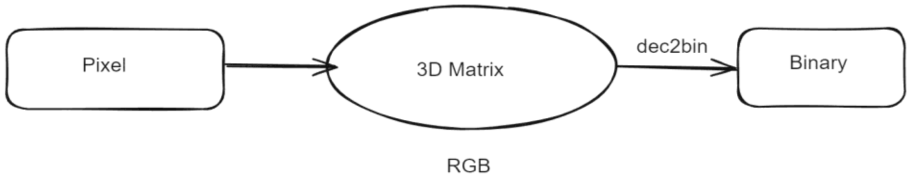


Fig. 4.2 Input image processing

2. **Key Generation**:

   The system generates a cryptographic key that will be used for encryption and decryption processes.

   Algorithm for key generation:

   Step 1: Input: User-provided key string

Step 2:

2.1 Convert the user-provided key string into a binary representation.

2.2 Perform bitwise operations (e.g., XOR) on the binary key to generate a key matrix with integer values.

2.3Ensure that the key matrix has dimensions compatible with the image to be encrypted.

Step 3: Output: Key matrix with integer values

Implementation: Utilize MATLAB to implement the key generation algorithm, ensuring compatibility with the chosen encryption method.

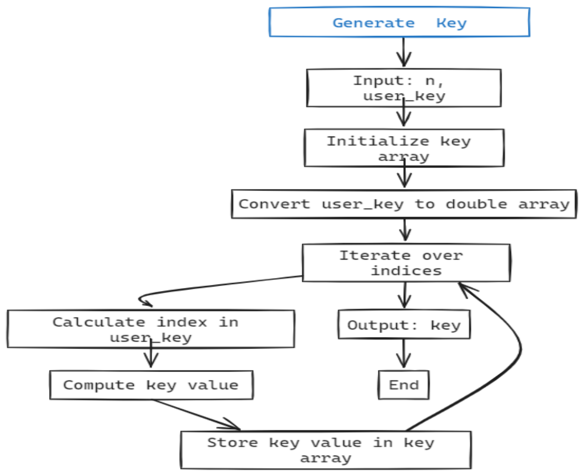


Fig 4.3 Implementation of Key Generation

3. **Encryption:**

The input image is encrypted using Visual Cryptography techniques, combined with additional encryption methods for enhanced security.

Algorithm for encryption:

Step1: Input: Original image, Key matrix

Step2:

2.1 Perform XOR operation between each pixel of the binary image and the corresponding pixel of the key matrix.

2.2 Partition the resulting binary image into shares, ensuring that each share reveals no discernible information about the original image.

Step3: Output: Encrypted shares

Implementation: Implement the encryption algorithm using MATLAB, incorporating XOR operation with the key and dividing the image into shares. Utilize Visual Cryptography principles to ensure that the encrypted shares reveal the original image when overlaid.

4. **Share Generation:**

The encrypted image is divided into multiple shares, each containing partial information.

Algorithm for share generation:

Step1: Input: Encrypted binary image

Step2:

2.1 Divide the encrypted binary image into distinct segments, each containing partial information.

2.2 Ensure that each shared image contains a portion of the encrypted data but reveals no information about the original image.

2.3 Save each share as a separate image file (e.g., share1.png, share2.png).

Step3: Output: Share images

Implementation: Utilize MATLAB to generate shares from the encrypted image, adhering to Visual Cryptography principles to ensure that individual shares reveal no discernible information about the original image.

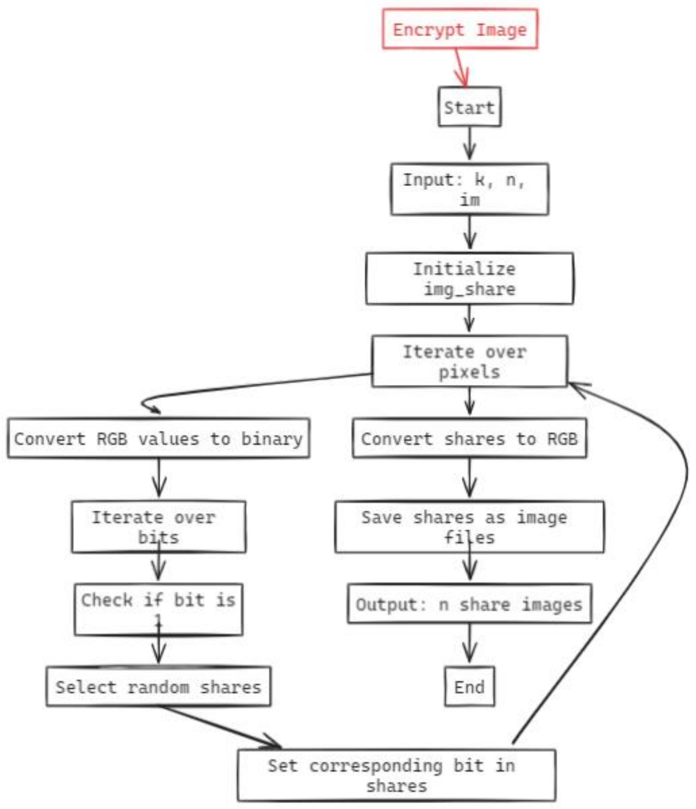


Fig 4.4 Process of Image Encryption

5. **Decryption:**

   To recover the original image, the shares generated during encryption are combined using specific decryption techniques.

   Algorithm for decryption:

   Step1: Input: Share images

   Step2:

   2.1Overlay the shared images on top of each other.

   2.2 Perform bitwise operations (e.g., XOR) between corresponding pixels of the shared images.

   2.3 Reconstruct the original binary image from the combined shares.

   2.4 Convert the binary image back to its original format (e.g., RGB).

   Step3: Output: Decrypted image

Implementation: Implement the decryption algorithm using MATLAB, ensuring that the shares are combined correctly to reconstruct the original image. Utilize Visual Cryptography principles to ensure that decryption is straightforward and does not require complex calculations.

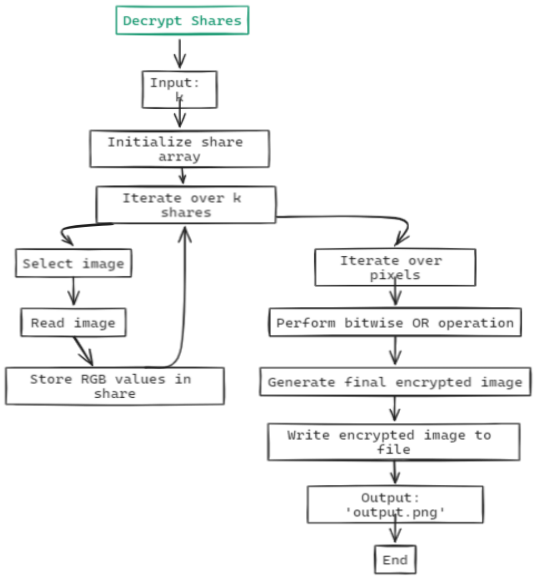


Fig 4.5 Image Decryption Steps

6. **Output Generation**:

   Once decryption is complete, the system generates the final output, which is the decrypted version of the original image.

   Implementation: Utilize MATLAB to generate the decrypted output image, ensuring that it matches the original input image. Save the output image in a suitable format for further analysis or use.

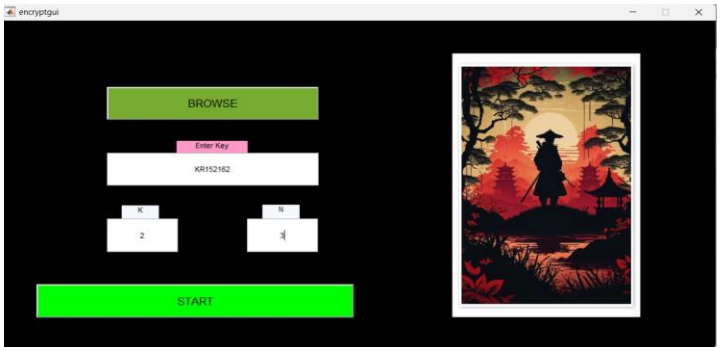
4.3 User Interface

Encryption GUI

The encryption GUI is designed to be intuitive and easy to navigate:

- Browse: Button to select the image to be encrypted.
- Enter Key: Field to input the encryption key.

- K (Shares Needed): Input field to specify the number of shares required for decryption.
- N (Total Shares): Input field to specify the total number of shares to be created.
- Start: Button to initiate the encryption process after all parameters are set.
- Display Area: Section to display the selected image.



Enter Key: KR152162

K: 2

N: 3

Fig 4.6 Encryption UI

Decryption GUI

The decryption GUI mirrors the encryption GUI in terms of layout and functionality:

- Enter Key: Field to input the decryption key.
- K (Shares Needed): Input field to specify the number of shares required for decryption.
- Start: Button to initiate the decryption process after all parameters are set.
- Display Area: Section to display the decrypted image.

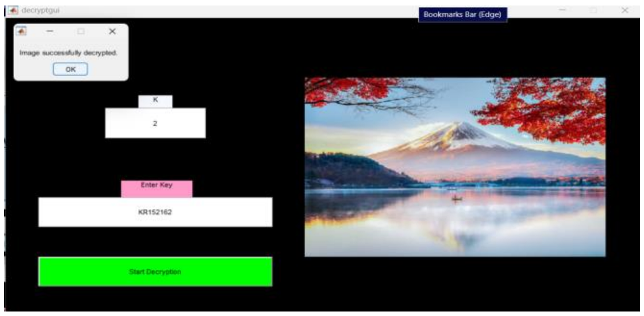


Image successfully decrypted.

K: 2

Enter Key: KR152162

Fig 4.7 Decryption UI

CHAPTER 5
TESTING AND RESULTS

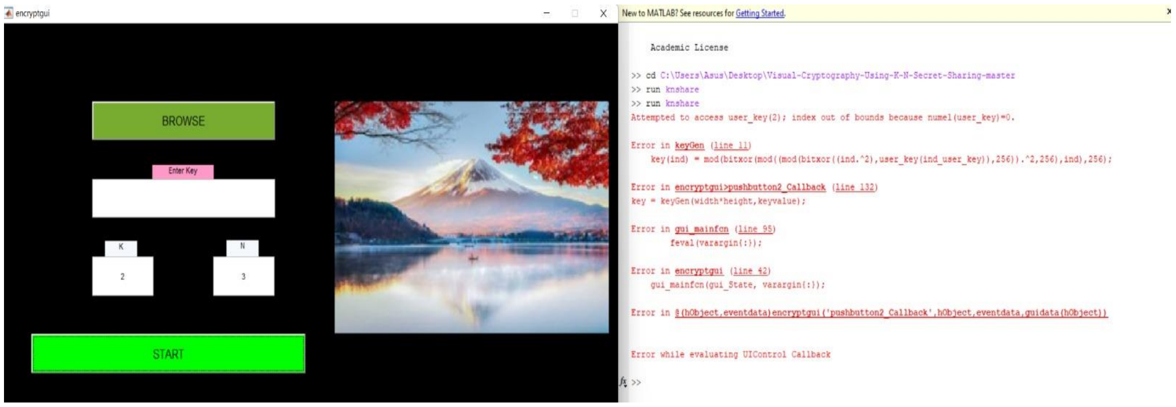
This chapter presents the testing methodology and results obtained from applying different combinations of shares in the visual cryptography scheme. The tests aimed to evaluate the impact of using various shares on the resulting image quality. Specifically, we focused on a threshold scheme with k (threshold) set to 2 and n (total shares) set to 3. The test cases involved different combinations of identical and distinct shares to observe their effects on the reconstructed image. The results highlight how the choice of shares influences the clarity and accuracy of the decrypted image. Through these experiments, we demonstrate the effectiveness of the visual cryptography scheme in preserving the original image quality while ensuring secure encryption and decryption processes.

5.1 Test Cases

TEST CASE FOR KEY GENERATON

TEST CASE 1: TEST WITH NO KEY/EMPTY KEY

If key is not given in encryption process, it results in an error.



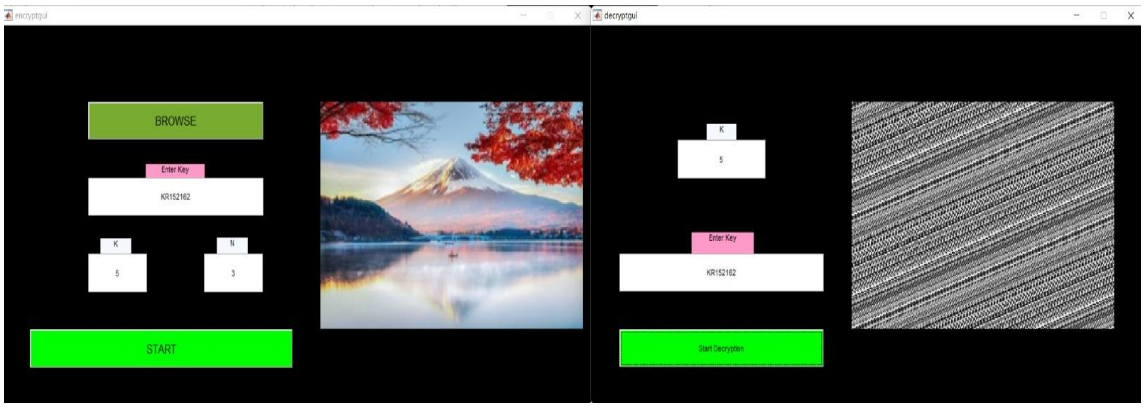
```
    Academic License

>> cd C:\Users\Asus\Desktop\Visual-Cryptography-Using-K-N-Secret-Sharing-master
>> run knshare
>> run knshare
Attempted to access user_key(2); index out of bounds because numel(user_key)=0.

Error in keyGen (line 11)
    key(ind) = mod(bitxor(mod((mod(bitxor((ind.^2),user_key(ind_user_key)),256)).^2,256),ind),256);

Error in encryptgui>pushbutton2_Callback (line 132)
key = keyGen(width*height,keyvalue);

Error in gui_mainfcn (line 95)
        feval(varargin{:});

Error in encryptgui (line 42)
    gui_mainfcn(gui_State, varargin{:});

Error in @(hObject,eventdata)encryptgui('pushbutton2_Callback',hObject,eventdata,guidata(hObject))


Error while evaluating UIControl Callback

>>
```

Fig. 5.1 Test Case with no key/empty key

TEST CASE FOR ENCRYPTION

TEST CASE 1: WHEN k IS GREATER THAN n

When the given k(5) value is greater than n(3) then it results in generation of 3 shares during encryption. But during decryption we are asked to select 5 shares to decrypt which resulted in reputation of shares and the output image is not same as original input image



Enter Key: KR152162

K: 5

N: 3

K: 5

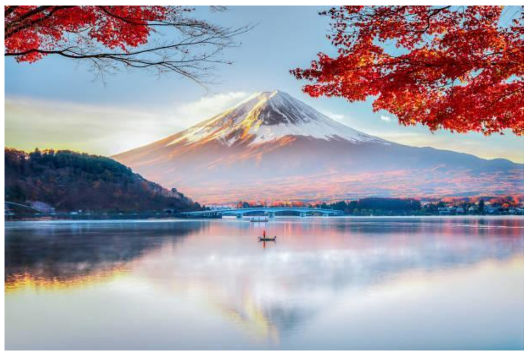
Enter Key: KR152162

Fig.5.2 Test Case for k > n

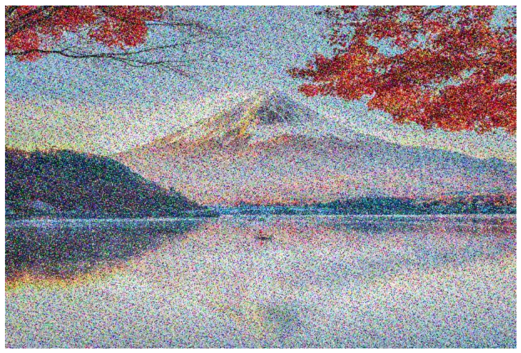
TEST CASE FOR DECRYPTION

The following test cases were conducted with k (threshold)set to 2 and n set to 3:

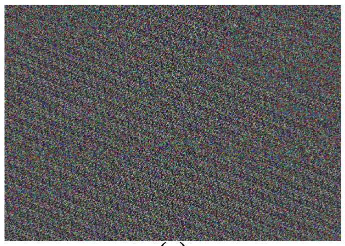
TEST CASE 1: IDENTICAL SHARES (Share 1 & Share 1):



(a)



(b)



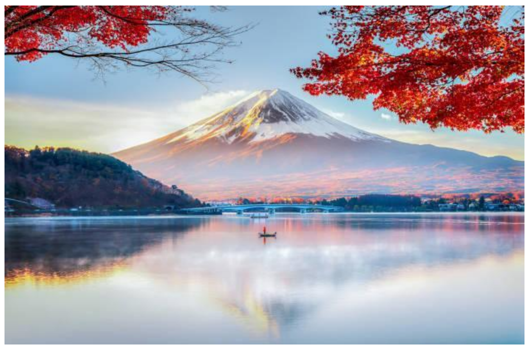
(c)

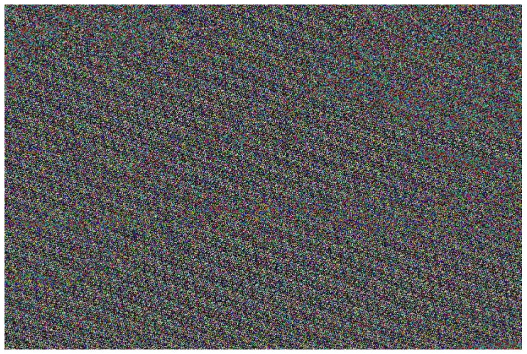
Fig 5.3 Test case using same shares (a) Input image (b) Output image (c) Share1

During decryption, when asked to select the shares, two same shares (Share 1 & Share 1) were given, which resulted in a blurry/dotted image. Using identical shares leads to all bits being set to 1 (due to OR with itself), resulting in a brighter and potentially saturated image with limited detail.

TEST CASE 2: DECRYPTION USING DIFFERENT SHARES (Share 1 & Share 2)

(a)


(b)


(c)

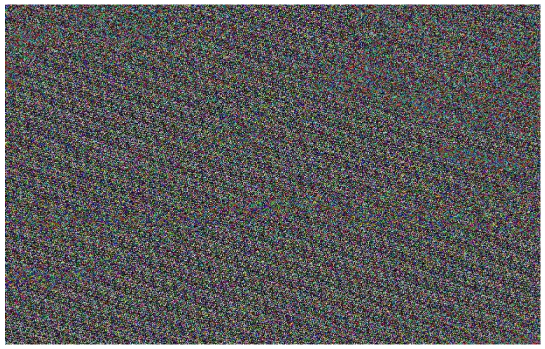

(d)

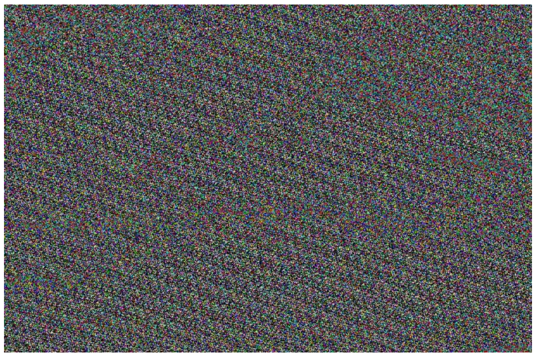
Fig 5.4 Test case using unique shares (a) Input image (b) Output image (c) Share1 (d) Share 2

During decryption, when asked to select the shares, two different shares (Share 1 & Share 2) were given, which produced the original image. Using different shares introduces some variation in the bitwise OR operation, leading to a less saturated and potentially more accurate appealing image.

TEST CASE 3: DECRYPTION USING DIFFERENT SHARES (Share 2 & Share 3):

(a)

(b)


(c)

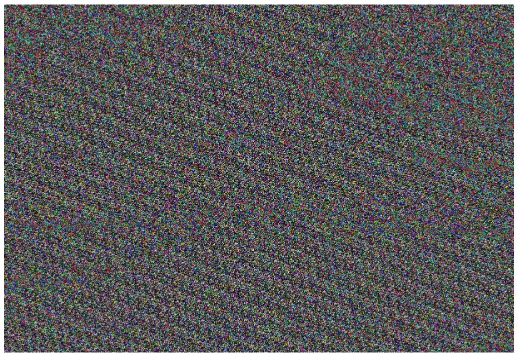
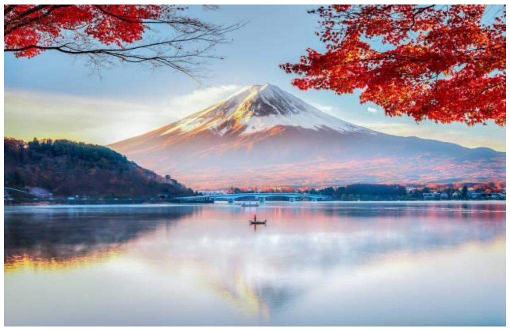

(d)

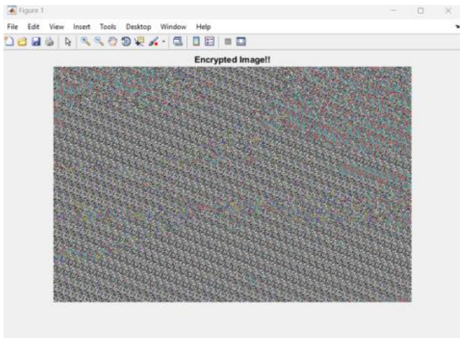
Fig 5.5 Test case using unique shares (a) Input image (b) Output image (c)Share 2 (d) Share 3

During decryption, when asked to select the shares, another combination of two different shares (Share 2 & Share 3) were given, which also produced the original image. Using different shares introduces some variation in the bitwise OR operation, leading to a less saturated and potentially more accurate appealing image.

Result:

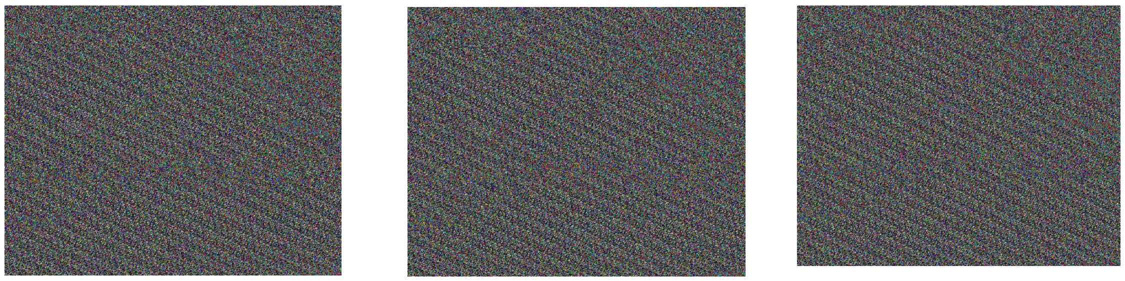
Encryption:


(a)



(b)

Fig 5.6 (a) Input Image (b) Encrypted Image



(a) (b) (c)

Fig 5.7 Shared images (a) share image 1 (b) share image 2 (c) share image 3

User Input Image: An original image was selected as the input for encryption

Encrypted Image: The output is considered the encrypted representation of the original image. This encrypted data protects the image content from unauthorized access.

Shares: These shares are essential for decryption and should be distributed securely to different locations. Recovering the original image requires a minimum of k shares (where k is less than n).

Decryption:

User Input: Decryption can be done using the user interface (UI) to select a set of k shares from the total n shares generated during encryption.

Decrypted Image: If the shares are selected correctly, then the output image should visually resemble the original image used for encryption.

CHAPTER 6
CONCLUSION

The “Secret image sharing using Shamir’s Secret Sharing algorithm” project has successfully achieved its objectives of implementing a secure and efficient encryption and decryption system for multiple images using Visual Cryptography techniques. the existing cryptographic methods often face challenges in securely transmitting and sharing sensitive information, especially when it comes to visual data such as images. Traditional encryption techniques may compromise image quality or require complex decryption processes. Moreover, the vulnerability of centralized key management systems poses a significant security risk. To address these flaws, our proposed system leverages visual cryptography, a novel approach that divides images into multiple shares, offering a decentralized and robust method for image encryption. By implementing Shamir's secret sharing algorithm and XOR operations, we ensure that the original image can only be reconstructed by combining a subset of shares and using the correct key. This enhances security and privacy while maintaining the integrity of the shared images.

Through this project, we successfully demonstrated the effectiveness and feasibility of Shamir’s Secret sharing algorithm in securing image transmission and sharing. By providing a decentralized and secure method for encrypting images, our system addresses the shortcomings of traditional cryptographic methods and offers a practical solution for protecting sensitive visual data in various applications.

6.1 Key Achievements

1. **Algorithm Implementation:** The project successfully implemented encryption and decryption algorithms for visual cryptography using MATLAB, incorporating XOR operations with a key before share generation. This approach has enhanced the security and randomness in the encryption process, ensuring data privacy and integrity.
2. **Share Generation and Image Reconstruction:** The developed system effectively divides the XORed image into shares using visual cryptography algorithms, and the

decryption algorithm reconstructs the original XORed image from a subset of shares, enabling secure and efficient image transmission.

3. **Enhanced Security:** The integration of key generation and random place selection algorithms has significantly enhanced the security and randomness in the encryption process, safeguarding against cryptographic attacks and ensuring the confidentiality of the encrypted shares.
4. **Feasibility and Effectiveness:** Through implementation and experimentation, the project has demonstrated the feasibility and effectiveness of the proposed visual cryptography scheme for secure image transmission and sharing. The encrypted shares maintain confidentiality and can only be decrypted with the correct combination of shares and the secret key.

6.2 Contributions and Future Implications

The project's contributions extend to the understanding and practical implementation of visual cryptography techniques for secure image transmission and sharing applications. The proposed system leverages visual cryptography techniques to address the limitations of traditional cryptographic methods, providing visual confirmation of encrypted data and eliminating the need for sharing decryption keys among parties, thereby enhancing data privacy, integrity, and security.

The successful completion of this project opens avenues for further research and development in the field of visual cryptography, encryption algorithms, and secure data transmission. Future implications include exploring methods to enhance the security of the encryption process, incorporating additional encryption techniques, optimizing share generation algorithms, and developing user-friendly software applications for automating the encryption and decryption processes.

REFERENCES

[1] Moni Naor and Adi Shamir, "Visual Cryptography," https://eprint.iacr.org/

[2] Visual Cryptography Explained | tsumarios' blog - Mario Raciti, https://github.com/topics/visual-cryptography

[3] GeeksforGeeks, "Introduction to XOR Operation," https://www.geeksforgeeks.org/tag/xor/

[4] Adi Shamir, "Visual Cryptography: Fair Verifiable Secret Sharing," (Shamir's original paper on Visual Cryptography)

[5] Lei, Ting; Gou, Tao; Peng, Dongmei; Sun, Jian (2016). "Visual Cryptography for Fingerprint Biometric Template Protection". IEEE Transactions on Information Forensics and Security. 11 (1): 12–24. doi:10.1109/TIFS.2015.2493223. (This paper discusses visual cryptography for protecting fingerprint biometric templates)

[6] Security Analysis of Visual Cryptography:

[7] Fridrich, Miroslav; Simoens, Jessica; Tuyls, Pim (2001). "Attacking Visual Cryptographic Schemes with Brightness and Contrast Modification". Lecture Notes in Computer Science. 2065: 137–154. doi:10.1007/3-540-44973-7_12. (This paper explores potential security weaknesses in VC schemes related to manipulating brightness and contrast)

[8] Li, Bo; Li, Jun; Huang, Wenbo (2018). "Cryptanalysis of a Visual Secret Sharing Scheme Based on Error Diffusion". Security and Communication Networks. 2018 (1): 1–8. doi:10.1155/2018/9217524. (This paper presents a cryptanalysis of a specific visual cryptography scheme)

APPENDIX

Encryptgui.m

```
function varargout = encryptgui(varargin)
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn',
@encryptgui_OpeningFcn, ...
                   'gui_OutputFcn',  @encryptgui_OutputFcn,
...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State,
varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end


function encryptgui_OpeningFcn(hObject, eventdata, handles,
varargin)
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);



% --- Outputs from this function are returned to the
command line.
function varargout = encryptgui_OutputFcn(hObject,
eventdata, handles)
% varargout  cell array for returning output args (see
VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)
```

```
% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)
global im;
global width;
global height;
X = uigetfile('*.jpg;*.bmp;*.tiff;*.ppm;*.pgm;*.png','pick
a jpg file');
im = imread(X);
[width, height, ~] = size(im);

axes(handles.axes1)
imshow(im);
guidata(hObject, handles);


function edit1_Callback(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)


% --- Executes during object creation, after setting all
properties.
function edit1_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end


% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
```

```
% handles    structure with handles and user data (see
GUIDATA)

global width;
global height;
global im;
k = get(handles.edit2,'String');
k = str2num(k);
n = get(handles.edit3,'String');
n = str2num(n);
keyvalue = get(handles.edit1,'String');
key = keyGen(width*height,keyvalue);
encrypted_im = imageProcess(im,key);
figure();
imshow(encrypted_im);
title('Encrypted Image!!');

kn_encrypt(k,n,encrypted_im);
guidata(hObject, handles);
msgbox('Shares generated.');

function edit2_Callback(hObject, eventdata, handles)
% hObject    handle to edit2 (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)

% Hints: get(hObject,'String') returns contents of edit2 as
text
%        str2double(get(hObject,'String')) returns contents
of edit2 as a double


% --- Executes during object creation, after setting all
properties.
function edit2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit2 (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    empty - handles not created until after all
CreateFcns called

% Hint: edit controls usually have a white background on
Windows.
%       See ISPC and COMPUTER.
```

```
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit3_Callback(hObject, eventdata, handles)
% hObject    handle to edit3 (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)

% Hints: get(hObject,'String') returns contents of edit3 as
text
%        str2double(get(hObject,'String')) returns contents
of edit3 as a double


% --- Executes during object creation, after setting all
properties.
function edit3_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit3 (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    empty - handles not created until after all
CreateFcns called

% Hint: edit controls usually have a white background on
Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

Decryptgui.m

```
function varargout = decryptgui(varargin)
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', 
@decryptgui_OpeningFcn, ...
                   'gui_OutputFcn',  @decryptgui_OutputFcn, 
...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, 
varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end


% --- Executes just before decryptgui is made visible.
function decryptgui_OpeningFcn(hObject, eventdata, handles, 
varargin)
handles.output = hObject;

% Update handles structure
guidata(hObject, handles)

% UIWAIT makes decryptgui wait for user response (see 
UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the 
command line.
function varargout = decryptgui_OutputFcn(hObject, 
eventdata, handles)
% varargout  cell array for returning output args (see 
VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version 
of MATLAB
```

```
% handles    structure with handles and user data (see
GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)

keyvalue = get(handles.edit1,'String');
k = get(handles.edit2,'String');
k = str2num(k);

[width height channel] = size(imread('share1.png'));
key = keyGen(width*height,keyvalue);
kn_decrypt(k);
figure();
imshow('output.png');
title('Merged k shares...');
output_image = imageProcess(imread('output.png'),key);
imwrite(output_image,'decryptedoutput.png','png');
axes(handles.axes1)
imshow('decryptedoutput.png');
msgbox('Image successfully decrypted.');
guidata(hObject, handles);


function edit1_Callback(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)

% Hints: get(hObject,'String') returns contents of edit1 as
text
%        str2double(get(hObject,'String')) returns contents
of edit1 as a double
```

```
% --- Executes during object creation, after setting all
properties.
function edit1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    empty - handles not created until after all
CreateFcns called

% Hint: edit controls usually have a white background on
Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end



function edit2_Callback(hObject, eventdata, handles)
% hObject    handle to edit2 (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)

%        str2double(get(hObject,'String')) returns contents
of edit2 as a double


% --- Executes during object creation, after setting all
properties.
function edit2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit2 (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    empty - handles not created until after all
CreateFcns called

%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

imageprocess.m

```
%Input :- ImgInp, key
% ImgInp = Input image which is to be encrypted
% key = the key matrix generated by keyGen

%Output :- Gives the encrypted image using the input key
function [ImgInp] = imageProcess(ImgInp,key)
[n m p] = size(ImgInp);
ind = 1;
for i = 1:n
    for j = 1:m
        for k = 1:p
            ImgInp(i,j,k) = bitxor(ImgInp(i,j,k),key(ind));
        end;
        ind = ind + 1;
    end;
end;
return;
```

keygeneration.m

```
%Input :- n,user_key
% n = int value which is area of image (width*height)
% user_key = string which is the user supplied key

%Output :- a key matrix with int values and length = n
function [key] = keyGen(n,user_key)
user_key = double(user_key);
key = zeros(n,1,'uint8');
for ind = 1 : n
    ind_user_key = mod(ind,size(user_key,2)) + 1;
    key(ind) =
mod(bitxor(mod((mod(bitxor((ind.^2),user_key(ind_user_key))
,256)).^2,256),ind),256);
end;
```

kn_decrypt.m

```
%Input :- K
% K = int value that is the number of shares avaliable with
user

%Output :- An image named 'output.png' that is made using
given k shares
%          and is encrypted
function [] = kn_encrypt(k);
im1 = imread('share1.png');

[im_width, im_height, channels] = size(im1);
share = zeros(k,im_width,im_height,3);
final = zeros(im_width, im_height, 3);

for i = 1:k
    X =
uigetfile('*.jpg;*.bmp;*.tiff;*.ppm;*.pgm;*.png','pick a
jpg file');
    im = imread(X);
    for j = 1:im_width
        for l = 1:im_height
            share(i,j,l,1) = im(j,l,1);
            share(i,j,l,2) = im(j,l,2);
            share(i,j,l,3) = im(j,l,3);
        end;
    end;
end;

for i = 1:k
    for j = 1:im_width
        for l = 1:im_height
            final(j,l,1) =
bitor(final(j,l,1),share(i,j,l,1));
            final(j,l,2) =
bitor(final(j,l,2),share(i,j,l,2));
            final(j,l,3) =
bitor(final(j,l,3),share(i,j,l,3));
        end;
    end;
end;

final = uint8(final);
imwrite(final, 'output.png', 'png');
```

kn_encrypt.m

```
%Input :- k,n,im
% K = int value of K selected by user
% N = int value of N selected by user
% im = input image (use .png format only)

%Output :- n different images that are saved as share1.png,
share2.png, etc..
function [] = kn_encrypt(k,n,im)
[im_width, im_height, channel] = size(im);

recons = n - k + 1;
img_share = uint8(zeros(n, im_width, im_height, channel*8));

for i = 1:im_width
    for j = 1:im_height
        PIX        =        [de2bi(im(i,j,1),8,'left-msb'),
de2bi(im(i,j,2),8,'left-msb'),      de2bi(im(i,j,3),8,'left-
msb')];
        for k = 1:24
            if(PIX(k) == 1)
                temp = randomPlace(n,recons);
                for l = 1:recons
                    img_share(temp(l),i,j,k) = 1;
                end;
            end;
        end;
    end;
end;

for i = 1:n
    red_share                                                 =
pixToRGB(squeeze(img_share(i,:,:,1:8)),im_width,im_height);
    green_share                                               =
pixToRGB(squeeze(img_share(i,:,:,9:16)),im_width,im_height)
;
    blue_share                                                =
pixToRGB(squeeze(img_share(i,:,:,17:24)),im_width,im_height
);

    ith_share = cat(3,red_share, green_share, blue_share);
    ith_share = uint8(ith_share);
    imwrite(ith_share,
strcat('share',num2str(i),'.png'),'png');
```

```
end;
```

knshare.m

```
function varargout = knshare(varargin)
% KNSHARE MATLAB code for knshare.fig
%       KNSHARE, by itself, creates a new KNSHARE or raises
the existing
%      singleton*.
%
%      H = KNSHARE returns the handle to a new KNSHARE or the
handle to
%      the existing singleton*.
%
%          KNSHARE('CALLBACK',hObject,eventData,handles,...)
calls the local
%       function named CALLBACK in KNSHARE.M with the given
input arguments.
%
%      KNSHARE('Property','Value',...) creates a new KNSHARE
or raises the
%     existing singleton*.  Starting from the left, property
value pairs are
%       applied to the GUI before knshare_OpeningFcn gets
called.  An
%       unrecognized property name or invalid value makes
property application
%     stop.  All inputs are passed to knshare_OpeningFcn via
varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI
allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help knshare

% Last Modified by GUIDE v2.5 22-Apr-2017 19:06:22

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn',    @knshare_OpeningFcn,
...
```

```
                   'gui_OutputFcn',  @knshare_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}]     =     gui_mainfcn(gui_State,
varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before knshare is made visible.
function  knshare_OpeningFcn(hObject,  eventdata,  handles,
varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of
MATLAB
%  handles      structure  with  handles  and  user  data  (see
GUIDATA)
% varargin   command line arguments to knshare (see VARARGIN)

% Choose default command line output for knshare
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes knshare wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command
line.
function varargout = knshare_OutputFcn(hObject, eventdata,
handles)
%  varargout   cell  array  for  returning  output  args  (see
VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of
MATLAB
```

```
% handles      structure with handles and user data (see
GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of
MATLAB
% handles      structure with handles and user data (see
GUIDATA)
run encryptgui


% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of
MATLAB
% handles      structure with handles and user data (see
GUIDATA)
run decryptgui
```

pixToRGB.m

```
%Input :- PIX,w,h
% PIX = a 3D vector containing bit values of a plane
% w = width
% h = height

%Output :- a 2D vector with bits changed to int, this image
corrosponds to
%          a graylevel plane
function [vec] = pixToRGB(PIX,w,h)
    vec = zeros(w,h);
    for i = 1:w
        for j = 1:h
            vec(i,j)  =  bi2de(squeeze(PIX(i,j,:))',2,'left-
msb');
        end;
    end;
```

```
return;
```

randomPlace.M

```
%Input :- n, recons
%Output :- An array that has 'recons' number of different
integer values
%          in the range of 1 to n
function [s] = randomPlace(n,recons)
    a = randperm(n);
    s = a(1:recons);
return;
```