

ASYMMETRIC ENCRYPTION WITH COMPRESSION-BASED STEGANOGRAPHY FOR SECURE DATA EXCHANGE

A Project Report submitted for the partial fulfillment of the requirement for the award of the
degree of

MASTER OF COMPUTER APPLICATIONS

SUBMITTED BY

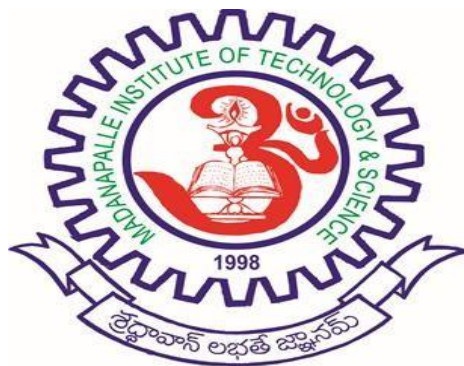
THANAKANTI VIJAY SIMHA REDDY
ROLL NO.: 23691F0004

Under the Guidance of

Dr. C. SIVARAJ

Associate Professor

Department of Computer Applications



DEPARTMENT OF COMPUTER APPLICATIONS

MADANAPALLE INSTITUTE OF TECHNOLOGY & SCIENCE

UGC AUTONOMOUS

(Approved by AICTE, New Delhi & Affiliated to JNTUA, Anathapuram)

www.mits.ac.in

2024-2025

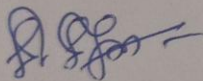
MADANAPALLE INSTITUTE OF TECHNOLOGY & SCIENCE
(UGC AUTONOMOUS)

(Approved by AICTE, New Delhi & Affiliated to JNTUA, Anathapuram)

DEPARTMENT OF COMPUTER APPLICATIONS

BONAFIDE CERTIFICATE

This is to certify that the project work entitled “**ASYMMETRIC ENCRYPTION WITH COMPRESSION-BASED STEGANOGRAPHY FOR SECURE DATA EXCHANGE**” is a Bonafide work carried out by **THANAKANTI VIJAY SIMHA REDDY** (Roll No.: 23691F0004), submitted in the partial fulfillment of the requirements for the award of the degree of Master of Computer Applications in Madanapalle Institute of Technology & Science, Madanapalle, affiliated to Jawaharlal Nehru Technological University Anantapur, Anathapuram the academic year 2024-2025.

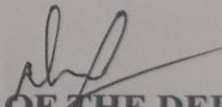


PROJECT GUIDE

Dr. C. SIVARAJ, Ph.D.,

Associate Professor

Department of Computer Applications



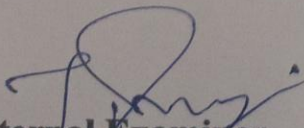
HEAD OF THE DEPARTMENT

Dr. N. Naveen Kumar, M.C.A., Ph.D.,

Professor

Department of Computer Applications

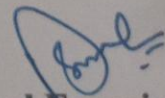
Submitted for viva voce examination held on 07/07/2025



Internal Examiner

Date:

7/7/25



External Examiner

Date:

7/7/25



MADANAPALLE INSTITUTE OF TECHNOLOGY & SCIENCE

(UGC-AUTONOMOUS INSTITUTION)

Affiliated to JNTUA, Ananthapuramu & Approved by AICTE, New Delhi

NAAC Accredited with A+ Grade, NIRF India Rankings 2021 - Band: 201-250 (Engg.)

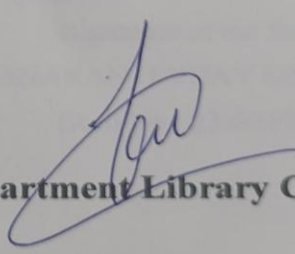
NBA Accredited - B.Tech. (CIVIL, CSE, ECE, EEE, MECH), MBA & MCA



Plagiarism Verification Certificate

This is to certify that the MCA Major Project entitled as “ASYMMETRIC ENCRYPTION WITH COMPRESSION-BASED STEGANOGRAPHY FOR SECURE DATA EXCHANGE” submitted by **Mr. THANAKANTI VIJAY SIMHA REDDY (Reg.No.23691F0004)** has been evaluated using **Anti-Plagiarism Software, TURNITIN** and based on the analysis report generated by the software, the similarity index is found to be **18 %**.

The following is the **TURNITIN** report for the Major Project consisting of **48** pages.

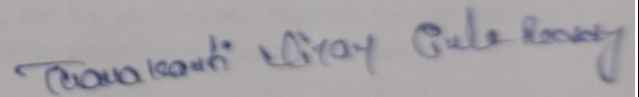

Department Library Coordinator

DECLARATION

I, THANAKANTI VIJAY SIMHA REDDY (Roll No:23691F0004) hereby declare that the project entitled "ASYMMETRIC ENCRYPTION WITH COMPRESSION-BASED STEGNOGRAPHY FOR SECURE DATA EXCHANGE" is done by me under the guidance of Dr. C. SIVARAJ, Ph.D., submitted in partial fulfillment of the requirements for the award of degree of Master of Computer Applications at MADANAPALLE INSTITUTE OF TECHNOLOGY & SCIENCE, Madanapalle affiliated to Jawaharlal Nehru Technological University Anantapur, during the academic year 2024-2025. This work has not been submitted by anybody towards the award of any degree.

Date: 07-07-2025

Place: Madanapalle



Signature of the Student

(THANAKANTI VIJAY SIMHA REDDY)

(Roll No.: 23691F0004)

ACKNOWLEDGEMENT

First, I must thank the almighty who granted me knowledge, wisdom, strength and courage to serve in this world and to carry out my project work in a successfully way.

Next, I have to thank my parents who encouraged me to undergo this MCA course and do this project in a proper way.

I express my sincere thanks to **Dr. N. Vijaya Bhaskar Chowdary, Ph.D.**, Secretary & Correspondent, Madanapalle Institute of Technology & Science for his continuous encouragement to wards practical education and constant support in all aspects which includes the provision of very good infrastructure facilities in the institute.

It is my duty to thank our Principal, **Dr. C. Yuvaraj M.E, Ph.D.**, Madanapalle Institute of technology & science, for his guidance and support at the time of my course and project.

It is my duty to thank Vice Principal Academics, **Dr. P. Ramanathan, Ph.D.**, for his continuous support though out our MCA career.

I express my sincere thanks to Vice Principal PG programs **Dr. R. Kalpana, Ph.D.**, for her constant support and encouragement during the project.

It is my foremost duty to thank Head of the Department **Dr. N. Naveen Kumar, M.C.A, Ph.D.** who gave me constant support during the project time and continuous encouragement to words the completion of the project successfully.

I thank my faculty guide **Dr. C. Sivaraj, M.C.A.**, for his continuous support by conducting periodical reviews and guidance until the completion of the project in as successful manner.

THANAKANTI VIJAY SIMHA REDDY

ABSTRACT

Providing security to data is a critical aspect for any industry. Cryptography and steganography are two pivotal techniques employed to protect sensitive information. Cryptography involves encrypting information so that only the intended recipient can decrypt and read the message. Steganography, on the other hand, transmits a secret message in a way that conceals its existence from potential intruders. An equally important aspect of information security is data compression, which not only makes data more secure but also easier to handle and transmit. The Rivest Shamir Adleman (RSA) encryption algorithm is used for its robust asymmetric encryption capabilities, ensuring that data remains confidential and accessible only to authorized recipients. Compressed encrypted data is then embedded into images using least significant bit (LSB) steganography, effectively concealing the data within a benign-looking image. This combination reduces physical storage space and transmission time over the Internet while providing a high level of security. The proposed system guarantees the encryption of data and its concealment from unauthorized access, addressing the dual challenges of data security and efficient transmission. Through this integrated approach, the system ensures that sensitive information remains protected and undetectable, thus significantly enhancing data security for firm. The proposed system clearly performs better than the existing one. It achieves a higher PSNR of 38.5 dB, meaning the image quality after data embedding is much clearer. With an SSIM of 0.98, the stego image closely resembles the original. The MSE is reduced to 2.1, showing that the data embedding is more accurate and less error-prone. Overall, the system's efficiency improves significantly from 75% to 92% making it both more secure and effective.

TABLE OF CONTENTS

CHAPTER	TITLE	PAGE NO
I	INTRODUCTION	
1.1	About the project	1-3
II	SYSTEM ANALYSIS	
2.1	Existing System	4
2.2	Proposed System	4
2.3	Hardware and Software Requirements	5
2.4	Feasibility Study	6
III	SYSTEM DESIGN	
3.1	Module Description	7-9
3.2	ER Diagram	9-10
3.3	UML Diagrams	10-18
IV	SYSTEM IMPLEMENTATION	
4.1	Language Selection	19-23
4.2	Screenshots	24-31
4.3	Sample Code	32-37
V	SYSTEM TESTING	
5.1	Testing	37-39
5.2	Test Objectives	40-42
5.3	Test Cases Passed	42
5.4	Test cases Failed	43
VI	CONCLUSION AND FUTURE ENCHANCEMENT	
6.1	Conclusion	44-45
6.2	Future Enhancements	45
	REFERENCES	46-48

LIST OF FIGURES

FIGURE NO.	TITLE	PAGE NO.
3.2.1	ER Diagram	9
3.3.1	Sender Use Case Diagram	11
3.3.1	Receiver Use Case Diagram	12
3.3.2	Class Diagram	13
3.3.3	Sequence Diagram	14
3.3.4	Activity Diagram	15
3.3.5	Component Diagram	16
3.3.6	Collaboration Diagram	17
3.3.7	Deployment Diagram	18
4.1.1.1	Python Architecture	20
4.3.1	Main window	24
4.3.2	Shows Textbox	25
4.3.3	Enter the Secret Message	26
4.3.4	After Click on Hide Data Button	27
4.3.5	Popup Message of Stego- Image Saved	28
4.3.6	Results of MSE PSNR and SSIM Values	29
4.3.7	Selecting the Stego-Image to Retrieve Data from The Image	30
4.3.8	After click on show data button the data will be shown on screen	31

LIST OF ABBREVAITIONS

S.NO.	ABBREVIATED TERM	EXPANSION
1	UML	Unified Modeling Language
2	ER Diagram	Entity Relationship Diagram
3	RSA	Rivest–Shamir–Adleman
4	LSB	Least Significant Bit
5	MSE	Mean Squared Error
6	PSNR	Peak Signal to Noise Ratio
7	SSIM	Structural Similarity Index

CHAPTER-I

INTRODUCTION

1.1 About the Project

In today's technology-driven world, we rely heavily on digital communication for everything from banking to healthcare and business operations. As we share personal and sensitive information over the internet, we need security to that data becomes critical. Unfortunately, traditional methods like simple encryption or basic hiding techniques are often not enough to keep our information safe from cyber threats. This project introduces a comprehensive and layered approach to securing data, using a combination of encryption, compression, and steganography.

The goal of this system is to securely transmit sensitive information such as passwords, and confidential documents. The uniqueness of the project lies in its integration of three proven techniques RSA encryption, Huffman compression, and Least Significant Bit (LSB) steganography. These techniques are used together to ensure the message is encrypted, compacted for efficiency, and hidden in a way that no one can even detect its existence.

The process begins with RSA encryption, which is a method of protecting the message using a pair of encrypting and decrypting. It checks person with the correct private key can unlock and read the message, even if someone intercepts it. Next, the Huffman coding technique is used to compress the encrypted message, making it smaller and faster to transmit while also reducing storage needs. Finally, the compressed message is hidden inside an image using LSB steganography, which replaces the least important bits of the image with parts of the hidden message. This method cleverly conceals the presence of data, making the image appear completely normal. To bring this concept to life, the system was developed using Python 3.9. The graphical interface was built using Tkinter, and additional Python libraries were used for encryption, compression, and image handling. The user interface is simple and easy to use even someone with no technical background can load an image, enter a message, and hide it within seconds. On the receiving side, the process is reversed: the image is loaded, the hidden data is extracted, decompressed, and finally decrypted to reveal the original message.

The system is built around important parts the sender module and the receiver module. One take care of encrypting the message, compressing it, and embedding it into the image. The receiver handles the extraction, decompression, and decryption. The system also includes a performance evaluation module that checks the quality of the image after embedding the message. This is done using three metrics: MSE, PSNR, and SSIM. These values tell us how much the image has changed and whether those changes are visible. In testing, the system consistently produced very low MSE values (indicating minimal change), PSNR values above 40 dB (suggesting high visual quality), and SSIM scores close to 1 (meaning the image looks virtually unchanged).

There are many advantages to this approach. Firstly, RSA encryption provides a strong layer of security, especially because it doesn't require a shared secret channel to work. Secondly, Huffman compression helps reduce data size, which makes the system more efficient and saves bandwidth. Thirdly, LSB steganography hides the message inside an image so well that it's nearly impossible to detect. The system is also highly scalable, supporting the transmission of text, images, and even video in the future. It's user-friendly, thanks to its clean interface, and versatile, suitable for use in fields like defense, healthcare, banking, and secure messaging.

Each part of the system encryption, compression, hiding, and extraction was tested individually (unit testing) to ensure it worked correctly. Then, all the parts were tested together (integration testing) to confirm they worked well as a complete system. Lastly, system testing was performed with real-life examples to make sure the entire process from start to finish was smooth and accurate. The results showed the system consistently succeeded in hiding and recovering messages without damaging the image or losing information.

The project is also feasible and practical. It uses free, open-source tools and works on standard hardware. It doesn't require expensive equipment or advanced skills to operate. From a technical perspective, it integrates well with existing software environments. Operationally, it requires little training and can easily fit into everyday workflows. Legally, the use of encryption and secure transmission aligns with data privacy laws and compliance requirements, making it a valid solution in professional settings.

The tools and libraries used in the project highlight its accessibility. The Tkinter library was

used to create the graphical interface, making it interactive and visually intuitive. The RSA and Huffman algorithms were implemented using custom Python modules, and the stegano.lsb library provided the ability to hide and extract messages from images with ease. This combination of tools ensures that the system is not only functional but also flexible and easy to maintain or extend in the future.

In conclusion, this project successfully addresses the need for secure, efficient, and invisible data transmission. By combining encryption, compression, and steganography, the system offers a powerful solution to protect data from unauthorized access and detection. It is lightweight, user-friendly, and delivers strong performance as proven by test results. With future upgrades such as support for audio and video files, integration of biometric authentication, or AI-based security enhancements, the system can be made even more robust. Overall, it has strong potential for real-world use in sectors that require high levels of confidentiality and secure communication.

CHAPTER-II

SYSTEM ANALYSIS

2.1 Existing System

Due to large size of the data being shared, the current method of online transmission is quite slow and inefficient. It also requires significantly more bandwidth, which can strain network resources. Additionally, there's always a risk that cyber attackers might intercept and access sensitive information during transmission. Since online data transfers don't guarantee complete security, there's a real concern about data breaches. On top of that, storing such large volumes of data demands extra physical storage space, which adds to the challenge.

Disadvantages

- No dual-layer protection; data vulnerable without both encryption and hiding.
- Lacks compression, causing slow transfer and high bandwidth or storage usage.
- No integrity checks; tampering undetected due to missing data verification.
- Limited scalability, needs secure key exchange; not suited for all formats.

2.2 Proposed System:

Proposed system uses a hybrid data compression approach that supports both lossy and lossless methods, ensuring flexibility and efficiency. To enhance security, it first encrypts the input data using the RSA (Rivest–Shamir–Adleman) algorithm. Compressing the encrypted data helps reduce its size, allowing for faster transmission and less storage usage, even across different storage platforms. Huffman coding is used to compress the encrypted content efficiently. Finally, the compressed, encrypted data is hidden within an image using the LSB steganography technique, making the hidden message nearly impossible to detect.

Advantages

- It combines encryption, compression, and steganography for strong, invisible data protection.
- It reduces message size, saving bandwidth and accelerating secure data transmission.
- It detects tampering using hashing, ensuring data integrity throughout communication.
- It supports multiple formats and users, providing a scalable and user-friendly solution.

2.3 Hardware and Software Requirements:

Hardware Requirements

Processor: Intel Core i3 (Tested on i5 and above)

RAM: Minimum 4 GB (Recommended: 8 GB for smoother performance)

Hard Disk: 256 GB or higher

System Type: 64-bit Operating System

Display: Standard resolution display (minimum 1024x768)

Software Requirements

Operating System: Windows 10 / Windows 11

Programming Language: Python 3.9

Editor/IDE: Visual Studio Code / PyCharm

2.4 Feasibility Study

2.4.1 Technical Feasibility

The project's technical viability establishes if it can be carried out with the resources and technology available today. Considerable elements include:

Compatibility: Ensuring the new system works with existing hardware and software.

Complexity: The level of sophistication of the steganography and encryption methods.

Resource Availability: Availability of technical expertise and necessary hardware.

Security: The strength of the encryption and steganography to withstand attacks.

2.4.2 Operational Feasibility

Operational feasibility looks at how well the proposed system will work in the existing operational structure. It includes:

User Acceptance: Will users adopt and properly use the new system?

Integration: How the system integrates into current workflows and processes.

Training: The training needed for users to effectively use the new system.

CHAPTER-III

SYSTEM DESIGN

3.1. Module Description:

3.1.1 Mean Squared Error (MSE)

This is the variation between the original picture data as found in Equation and the compressed image data. Its primary purpose is to assess our image's quality. It should be as small as feasible if it is zero, the compressed and original images are comparable and the method is known as lossless image compression.

$$\text{MSE} = \frac{1}{M \times N} \sum_{X=1}^M \sum_{Y=1}^N (f(X, Y) - f'(X, Y))^2$$

3.1.2 Peak Signal to Noise Ratio (PSNR)

This represents the relationship between the signal's intensity and the noise present in it. Everything is reliant on how good the picture is. The visual quality increases with a larger PSNR. It is based on the chosen image's MSE. Equation provides a mathematical expression for the PSNR, which is high when there is little discrepancy between the two pictures and excellent image quality

$$\text{PSNR} = 10 \log_{10}(\text{MAX}^2 / \text{MSE})$$

Where the highest pixel intensity in the ideal image is represented by MAX2, which is 2552. PSNR will be used to assess the steganography's quality [2], [8], and [10]. As can be shown in the testing findings, the Stego-Image has strong cognition, as demonstrated by the greater PSNR results and the achievement of more than 40 db.

3.1.3 Structural Similarity Index Measure (SSIM)

The SSIM is used to measure the perceived difference between two identical-looking images. It doesn't necessarily indicate which image is better in quality.

$$SSIM(x, y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_x\sigma_y + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)}$$

Instead, it helps us understand which image is likely to be the "original" and which one has been altered or processed, such as through data compression [30]. The SSIM index is defined by the following equation.

The values of μ_x and μ_y are calculated by analyzing the brightness of each image in both the horizontal (x) and vertical (y) directions. However, these constants are often ignored and set to zero, while still following the guidelines of the global quality index. As a result, the closer the SSIM value is to 1, the better the image quality.

Sender

The sender module typically includes the following components:

- **Encryption module:** This module encrypts the data using asymmetric encryption techniques, ensuring the data is received by only receiver.
- **Compression module:** This module compresses the data to reduce its size, making it easier to transmit and store.
- **Steganography module:** This module hides the encrypted data within other files, such as images or documents, using compression-based steganography techniques.
- **Data transmission module:** This module is responsible for transmitting the encrypted and steganographic-ally hidden data to the intended recipient over a secure network.
- **User interface module:** This module provides a friendly interface for the sender to input and send the data securely.

Receiver

The receiver module typically includes the following components:

- **Data reception module:** This module receives the encrypted and steganographic-ally hidden data transmitted by the sender module.

- **Decryption module:** This module decrypts the data using the appropriate private key.
- **Decompression module:** This module decompresses the data to its original size.
- **Detection module:** This module detects whether the received data contains any hidden data using steganalysis techniques.

3.2 ER Diagram

An Entity–Relationship (ER) model is a way to conceptually design the structure of a database. It's usually shown through an ER Diagram, which gives a visual representation of how different data entities are related. The main parts of an ER model are entity sets and relationship sets. An entity set is a collection of similar objects that have the same characteristics, known as attributes. These attributes describe the features of each entity. In database terms, an entity typically refers to a table, and the attributes are like the columns of that table. The ER diagram helps show how these tables (or entities) are linked together. This helps in understanding the complete logical design of the database before actual implementation.

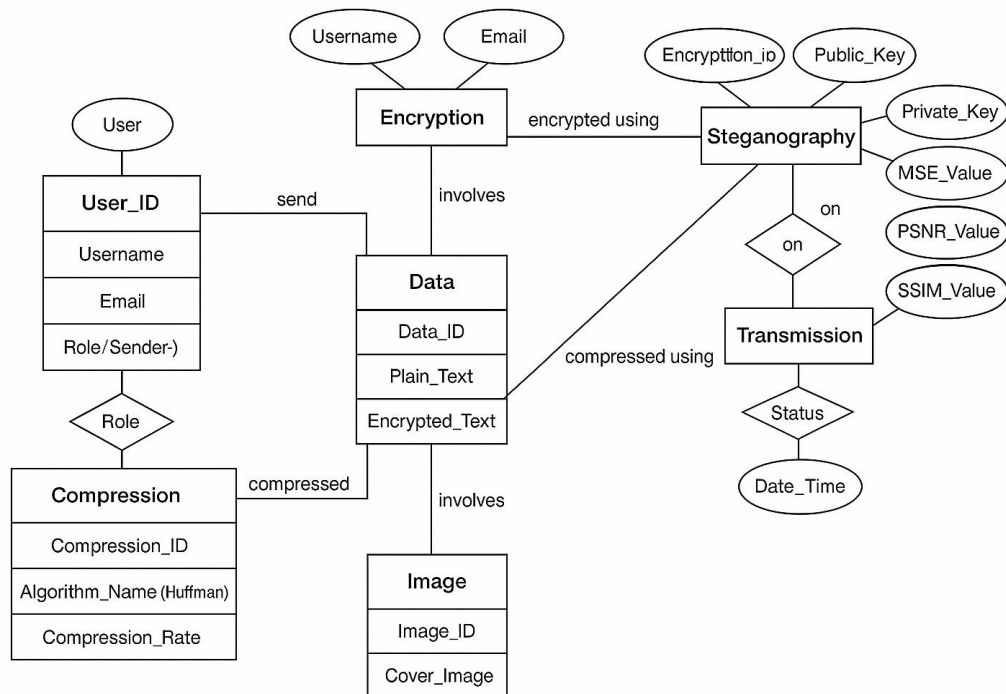


Fig.3.2.1:ER Diagram

The ER (Entity-Relationship) diagram outlines the design of a secure data transmission system that brings together encryption, compression, and steganography to safeguard sensitive information. It starts with the User entity, which holds basic user details like user ID, name, email, and role (such as sender). Users send data that gets captured in the Data entity, where both the original message (plain text) and the encrypted version are stored.

The encryption process is managed by the Encryption entity, which uses RSA encryption to protect the data, storing important parameters like encryption ID and public/private keys. After encryption, the data moves to the Compression module. Here, the system uses Huffman Encoding a lossless compression technique to reduce the size of the encrypted data, and it logs compression details like the algorithm name and compression rate.

Next, the Steganography module hides the compressed, encrypted data within an image. This process links the data to an image stored in the Image entity. Once hidden, the image is ready to be transmitted. The Transmission entity monitors this process, tracking the transmission status, time, and performance indicators such as PSNR, SSIM, and MSE, which help evaluate the quality and integrity of the stego-image.

Overall, this ER model reflects a well-organized system where each module plays a key role in ensuring that data is securely encrypted, efficiently compressed, and invisibly embedded resulting in a robust and safe communication framework.

3.3 UML Diagrams

UML, or Unified Modeling Language, is a standardized way of modeling that's widely used in object-oriented software development. It was created and is maintained by the Object Management Group (OMG). The main goal of UML is to offer a common method for visualizing and designing both the structure and behavior of object-oriented software systems.

At present, UML includes components: a meta-model that defines its rules and structure, and a notation system that represents the models visually. In the future, UML might also incorporate or link to a specific development methodology.

UML serves as a standard language for defining, visualizing, constructing, and documenting different components of a software system. It's also useful for modelling business processes and other systems outside of software development.

UML brings together a set of proven engineering practices that are effective in designing large, complex systems. It primarily uses graphical symbols to describe the architecture and design of a software application.

Main Goals of UML Design:

- To offer a ready-to-use and expressive visual modelling language for creating and sharing detailed models.
- To support extensions and customization of core elements.
- To remain independent of any specific programming language or development process.
- To provide a formal understanding of the modelling language.
- To support advanced development concepts such as components, patterns, frameworks, and collaborations.

3.3.1 Use Case Diagram:

A Use Case Diagram in Unified Modeling Language (UML) is a type of behavioral diagram created through use-case analysis. Its main purpose is to visually show what the system does in a clear and easy-to-understand way. The diagram includes actors (the users or other systems), their goals (represented as use cases), and how these use cases are connected or depend on each other. The key idea behind a use case diagram is to show which actions or operations the system performs for each actor. It helps illustrate what roles the actors play in the system and how they interact with the system's features.

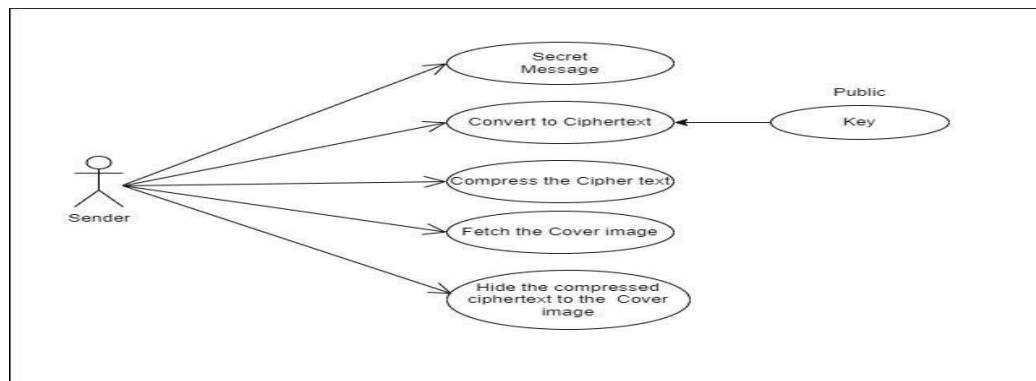


Fig.3.3.1: Sender Use Case Diagram

The Fig.3.3.1 shows the key steps carried out by the sender in a secure data transmission system. First, the sender creates a secret message that needs protection. This message is then encrypted

using a public key to generate unreadable ciphertext. To make the data more compact and easier to hide, the ciphertext is compressed. After that, the sender selects a suitable cover image to act as a hiding medium. Finally, the compressed ciphertext is converted to using steganography, ensuring the message remains hidden during transmission.

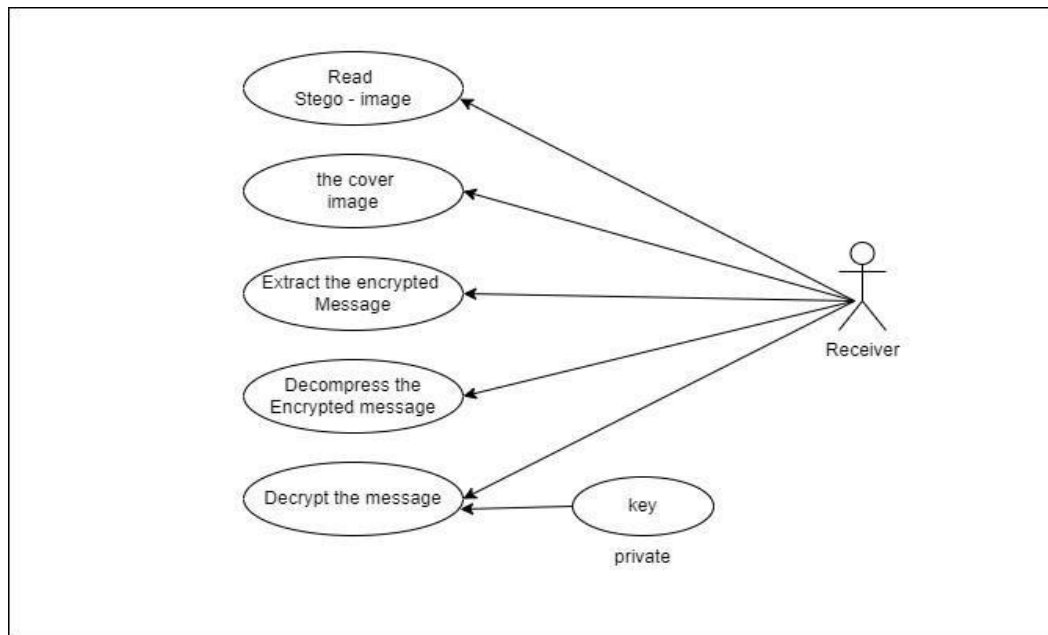


Fig.3.3.1: Receiver Use Case Diagram

The Fig.3.3.1 use case diagram explains the role of the receiver in retrieving securely hidden data. The process starts when the receiver opens the stego-image, which secretly holds the encrypted message. From this image, the hidden data is extracted and then decompressed to return it to its original encrypted format. Using their private key, the receiver decrypts the message, allowing them to read the original content. This step-by-step approach ensures the message stays protected and hidden until it reaches the right person.

3.3.2 Class Diagram

In software engineering, a class diagram in Unified Modeling Language (UML) is a type of static structure diagram that helps describe how a system is built. It shows the system's classes, their attributes, the methods (or operations) they perform, and how these classes are connected to each other. This diagram makes it easier to understand which class contains what kind of data or functionality within the system.

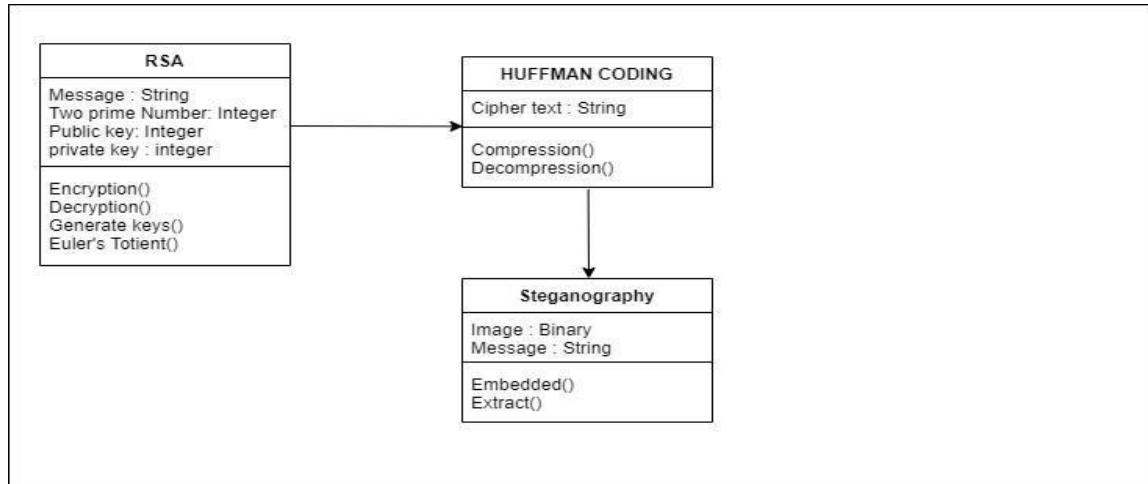


Fig.3.3.2. Class Diagram

The Fig.3.3.2 illustrates a secure data transfer system that combines RSA encryption, Huffman coding, and steganography to ensure the confidentiality and efficiency of data transmission. The process begins with the RSA class, which is responsible for generating encryption keys and performing both encryption and decryption using public and private keys. After the message is encrypted, it is forwarded to the Huffman Coding class, where the encrypted text is compressed to minimize its size. This compressed cipher text is then passed to the Steganography class, which embeds the message into an image, effectively concealing the data for secure transmission. Each class plays a distinct role, with defined attributes and methods that collectively work to enhance the security and effectiveness of sensitive data handling.

3.3.3 Sequence Diagram

In this sequence diagram, there are two primary classes: User and System. The user begins by selecting a dataset and uploading it to the system. Once uploaded, the user views the pre-processed dataset, while the system handles the data preprocessing. Then, the user selects an algorithm via the system, and the system proceeds to split the data into training and testing sets. After the system trains the data, the user applies the trained model to the dataset through the system. The system continues to train the data and model, the user then tests and classifies the data, and finally, the predicted results are generated and viewed by the user.

A sequence diagram includes a collection of objects, each represented by a lifeline, along with the messages exchanged between them over time. It displays the order of message flow and may also illustrate control structures between the objects.

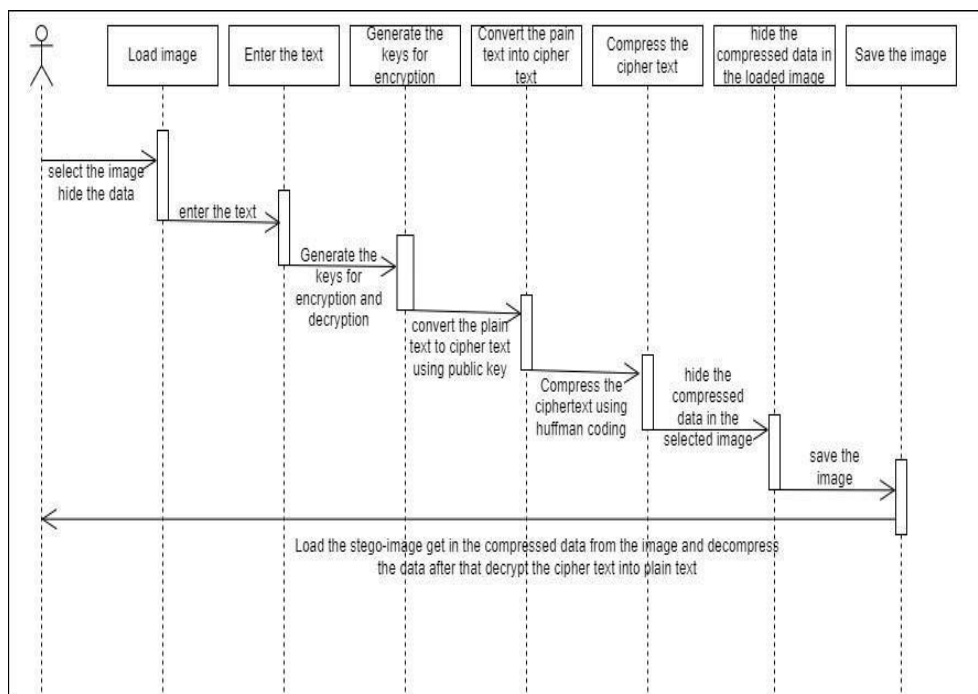


Fig.3.3.3: Sequence Diagram

The Fig.3.3.3 represents the complete workflow of securely hiding data by combining encryption, compression, and steganography techniques. The process begins with the user selecting an image and entering the text that needs to be hidden. Encryption keys are then generated, and the entered plain text is encrypted using the RSA algorithm. Once encrypted, the

cipher text is compressed using Huffman coding to reduce its size for efficient storage. This compressed cipher text is then embedded into the chosen image through steganography, ensuring the message remains hidden from unauthorized access. The final step involves saving the stego-image, which can later be used to extract, decompress, and decrypt the hidden data back into its original form.

3.3.4 Activity Diagram:

Activity diagrams are visual tools that show the flow of activities and actions in a sequence, including decisions, loops, and parallel tasks. In Unified Modeling Language (UML), activity diagrams are used to clearly represent detailed business processes and how different parts of a system operate step by step. They help highlight the overall flow of control within the system..

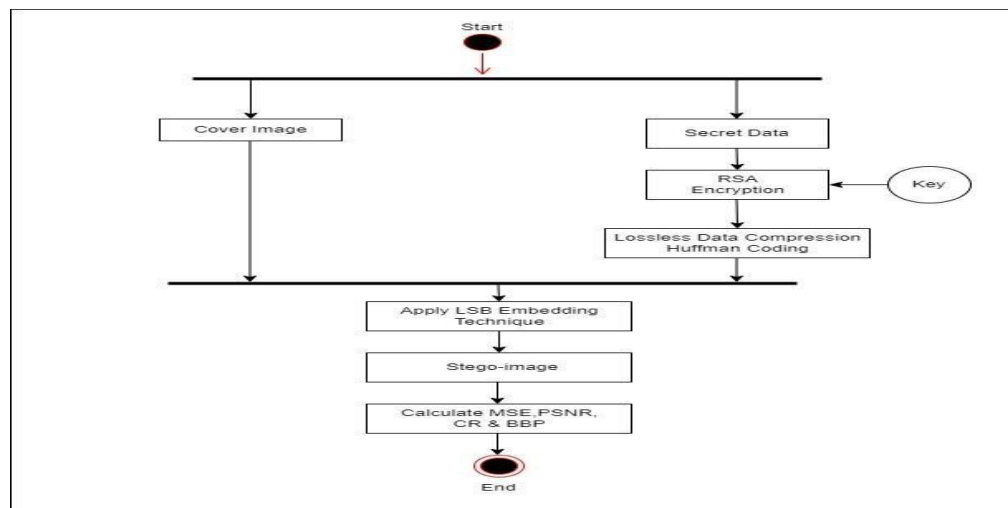


Fig.3.3.4: Activity Diagram

The Fig.3.3.4 illustrates the process of securely hiding data using a combination of cryptographic and steganographic techniques. The procedure starts with the encryption of sensitive information using the RSA algorithm, ensuring that the data is protected from unauthorized access. After encryption, the data undergoes lossless compression using Huffman coding to reduce its size without compromising quality. Meanwhile, a cover image is selected to serve as the carrier for the hidden information. The compressed, encrypted data is then embedded into the cover image using the LSB embedding technique, creating what is known as a stego-

image. To assess the effectiveness and quality of the data hiding process, key performance metrics such as MSE, PSNR, CR, and BBP are calculated.

3.3.5 Component Diagram:

A component diagram, also known as a UML component diagram, shows how the physical parts of a system are organized and linked together. These diagrams are mainly used to represent the system's implementation details and to make sure that all necessary features and functions are included during the development process.

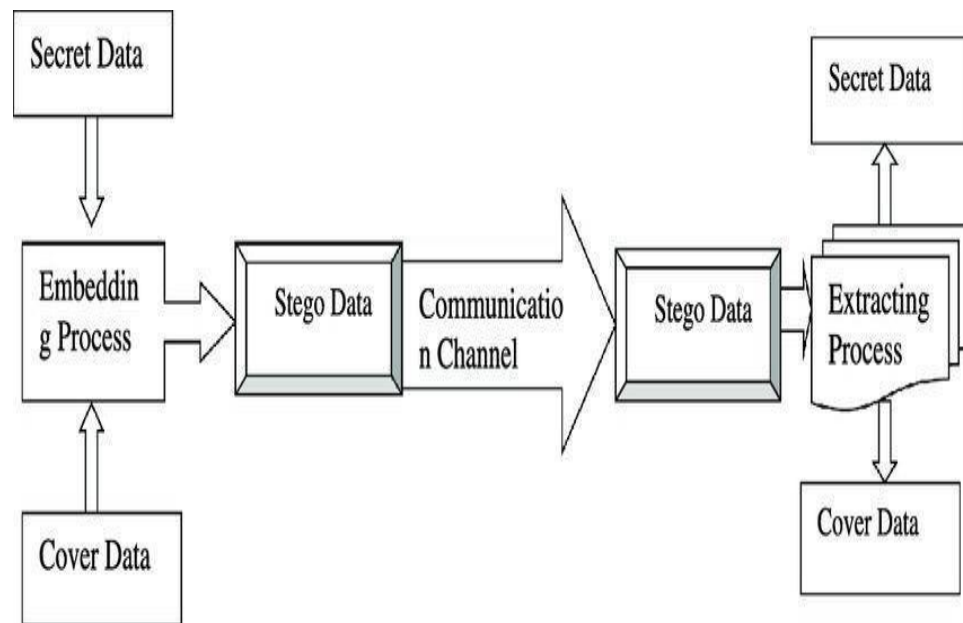


Fig.3.3.5: Component Diagram

The Fig.3.3.5 illustrates the fundamental concept of data hiding and retrieval using steganography. Initially, secret information is merged with cover data through an embedding process, resulting in stego data that conceals the hidden message. This stego data is then transmitted over a communication channel to the intended recipient. Upon receiving the stego data, an extraction process is carried out to separate and recover the original secret information while preserving the cover data. This method provides a secure way to transmit confidential information without drawing attention to its presence, ensuring privacy and protection against unauthorized access.

3.3.6 Collaboration Diagram:

In a collaboration diagram, the sequence of method calls is shown using numbered steps that indicate the order in which each method is called. To explain this better, we've used the same order management system example. The method calls in a collaboration diagram are much like those in a sequence diagram. However, the main difference is that a sequence diagram highlights the order of interactions over time, while a collaboration diagram also shows how the objects are organized and how they work together.

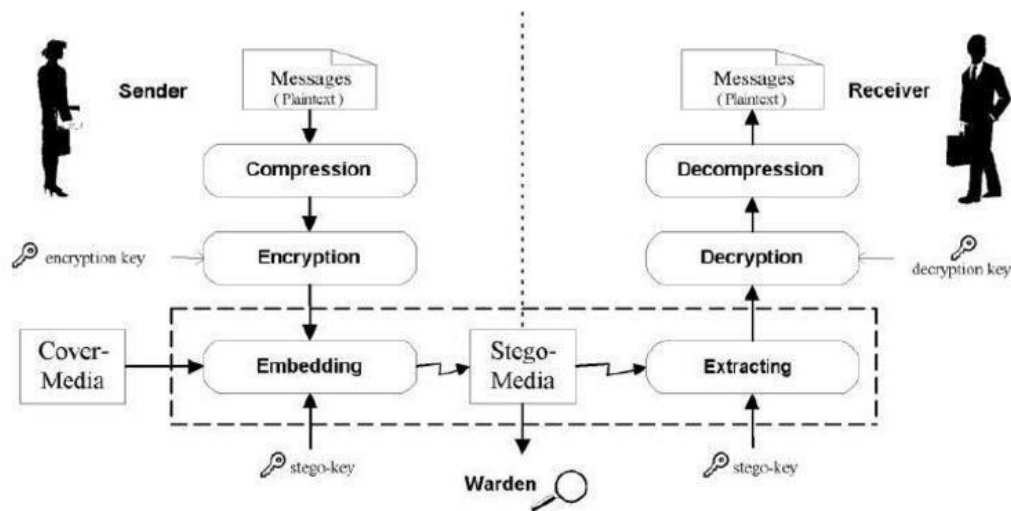


Fig.3.3.6: Collaboration Diagram

The Fig.3.3.6 number help indicate the sequence in which actions or method calls happen during the interaction between objects.

This approach is similar to what's used in a sequence diagram, where the flow of method calls is also shown. However, there's one main difference: while a sequence diagram focuses more on the timeline of events, a collaboration diagram highlights how objects are structured and interact with each other. So, while both diagrams show method calls, the collaboration diagram puts more emphasis on the roles and relationships between objects.

3.3.7 Deployment Diagram:

A deployment diagram shows how a system is physically set up and where its different parts are deployed. It's closely related to the component diagram, as it shows where those components are actually placed in the system. The diagram includes nodes, which represent the physical hardware devices—like servers or computers—that are used to host or run the application.

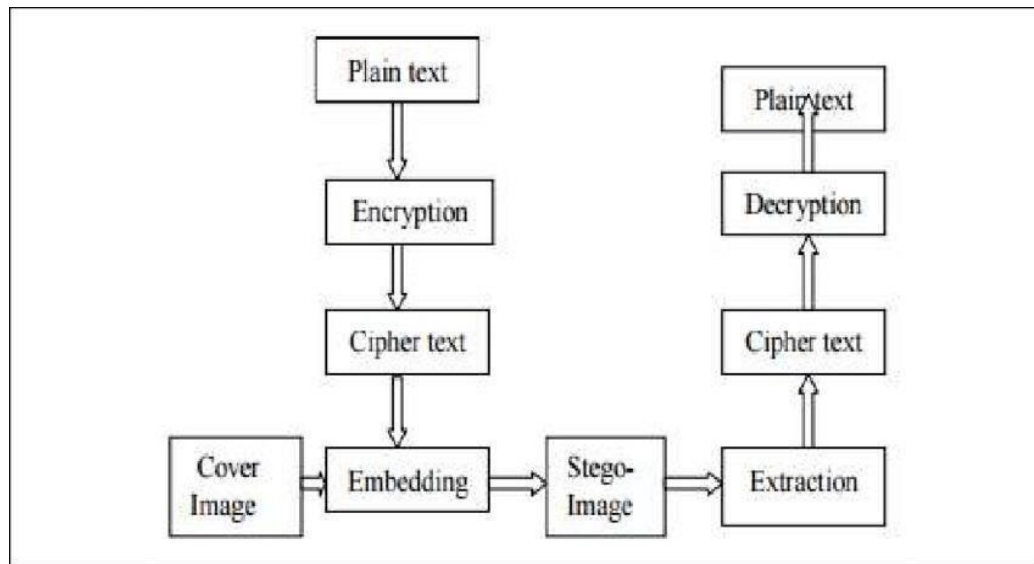


Fig.3.3.7: Deployment Diagram

The Fig.3.3.7 shows how a system is set up from a hardware or infrastructure point of view. It gives a clear picture of how different parts of the system (called components) are deployed in the real world. In these diagrams, you'll see nodes, which represent the physical hardware like servers or devices used to install and run the applications.

In simple terms, a deployment diagram helps us understand where and how the software is physically hosted and how everything is connected.

CHAPTER-IV

SYSTEM IMPLEMENTATION

4.1 Language Selection:

4.1.1 Introduction to Python

What is a Script?

A script refers to a file containing Python code, and it can be written using basic text editors like Notepad, or nearly any other word processing tool.

Difference Between a Script and a Program Script?

Until now, the focus has been on Python's interactive mode, where you type code and get immediate results. This feature is helpful for quick execution and testing of small code snippets. A script is essentially a plain text file with Python statements. Once written, it can be run multiple times without the need to retype the code. Scripts Are Editable. One of the biggest advantages of scripts is that they can be easily modified using a text editor. You can create different versions by changing parts of the code, making it simple to build variations of your program with minimal effort.

Scripts vs Application Code

Scripts are often separate from the core application logic, which is typically developed in another language. Scripts are usually interpreted, while applications they control are often compiled into native machine code.

What is a Program?

A program is a compiled and executable version of source code that the computer can run directly. The human-readable version is the source code, which gets converted into machine-readable instructions.

What is Python?

You might be wondering what Python is. Whether you're just starting out or you've heard of other languages like C, C++, C#, or Java, Python stands out for its simplicity and readability. This section explains why Python is a powerful and accessible programming language.

Python Concepts

If you're not interested in the internal details, feel free to skip ahead. Otherwise, here's why Python is an excellent language especially for beginners and how it supports both simple and complex application development.

- Python is an open-source, general-purpose language.
- Supports object-oriented, procedural, and functional programming styles.
- Interfaces easily with C, ObjC, Java, and Fortran.
- Can connect with C++ using tools like SWIG.
- Offers a powerful interactive environment.

Python's Key Features

- High-Level and Interpreted: Python code runs directly without needing compilation, similar to PERL and PHP.
- Interactive: You can write and test Python code directly in the interpreter prompt.
- Object-Oriented: Python allows you to build programs using classes and objects.
- Beginner-Friendly: Python is ideal for those new to programming and can be used to create everything from simple scripts to web browsers and games.

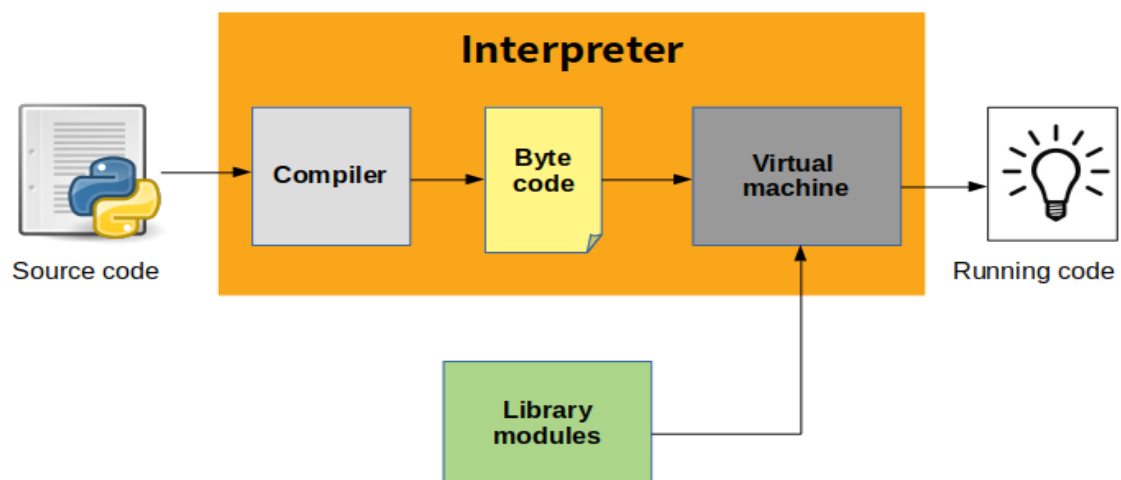


Fig.4.1.1.1:Python Architecture

Standard Data Types

Data Types in Python

Data stored in memory can have different forms. For instance, a person's age might be stored as a number, while their address might be stored using letters and numbers. Python offers several built-in **data types** that define how data can be handled and how it's stored.

Python's Five Standard Data Types

- Numbers
- String
- List
- Tuple
- Dictionary

Python Numbers

Numeric data types are used to store numbers. These number objects are created when a value is assigned to a variable.

Python Strings

A string is a continuous set of characters enclosed in quotation marks. Python supports both single and double quotes for strings. Parts of a string can be extracted using the slice operator ([and [:]), with indexing starting from 0 at the beginning and -1 from the end.

Lists are one of Python's most flexible data types. A list includes items separated by commas and enclosed in square brackets []. Unlike C arrays, a Python list can contain items of **different types**. You can use the slice operator to access items, just like in strings. Lists also support the + operator for concatenation and the * operator for repetition.

Python Tuples

Tuples are another form of sequence similar to lists. A tuple is made up of values separated by commas and enclosed in **parentheses** (). Unlike lists, tuples are **immutable**, meaning their size and contents can't be changed. While lists use square brackets, tuples use parentheses, making them like **read-only lists**.

Python Dictionary

Dictionaries in Python function like hash tables or associative arrays. They store **key-value pairs** and work similarly to hashes in Perl.

Keys are typically numbers or strings, and values can be any type of Python object. Dictionaries

are defined with **curly braces** {}, and data can be accessed or assigned using square brackets [].

Python Modes

Python supports two main modes of operation: **normal mode** and **interactive mode**.

- In normal mode, you run a .py script through the Python interpreter.
- In interactive mode, you can input code line by line and get immediate feedback, while keeping earlier lines active in memory.

Popular Python Libraries

Here are some widely used libraries in Python:

- **Requests:** A popular HTTP library developed by Kenneth Reitz; essential for web requests.
- **NumPy:** Offers high-performance operations for numerical data. It provides support for **multidimensional arrays**, linear algebra, and more.
- Works like MATLAB in Python
- Handles arrays efficiently
- Supports 2D, multi-dimensional arrays, etc.
- **Matplotlib:** A powerful library for creating high-quality **graphs and plots**.

Python Classes and Objects

In Python, **classes and objects** allow you to implement **object-oriented programming**. Classes define blueprints for objects, while objects are individual instances that hold data and behaviour defined by the class.

Python modules

TKINTER:

Tkinter is a built-in Python library used to create graphical user interfaces (GUIs) with components like buttons, text fields, labels, and menus. It allows the development of an intuitive GUI for a hybrid cryptography system using Vigenère and Polybius ciphers. Users can input plaintext, keys, and choose between encryption and decryption using simple widgets. The GUI displays both ciphertext and decrypted text, enhancing usability without coding knowledge. It also

provides visual feedback on system status and potential errors during operation.

LSB

The `lsb` module, which is a component of the steganography package, may be referred to as the "lsb" package when used with Python. This module offers routines that use least significant bit (LSB) steganography to conceal and extract data from digital photographs. By changing the least significant bit of each pixel in a picture, LSB steganography conceals data. Since the LSB modifications are frequently undetectable to the human sight, this may be done without noticeably changing the look of the image. methods for encoding data into an image using LSB steganography and methods for decoding data from an `LSencoded` picture are provided by the `lsb` module in the steganography package.

HUFFMAN

The Huffman coding technique, which compresses data by giving each symbol in a given data set a variable-length code, is implemented simply and effectively by the Huffman module in Python. Each code has a length that is decided by how frequently the associated symbol occurs in the data collection; symbols that appear more frequently are given shorter codes than symbols that occur less frequently. In some cases, Huffman coding can achieve very high compression ratios, particularly when the data has a high frequency of often occurring symbols combined with a large number of rarely occurring symbols. With the use of m's code completion and refactoring capabilities, we can produce well-written, reusable code.

4.2 SCREEN SHOTS

The figure depicts the main window of the application which is displayed through executing the application file.

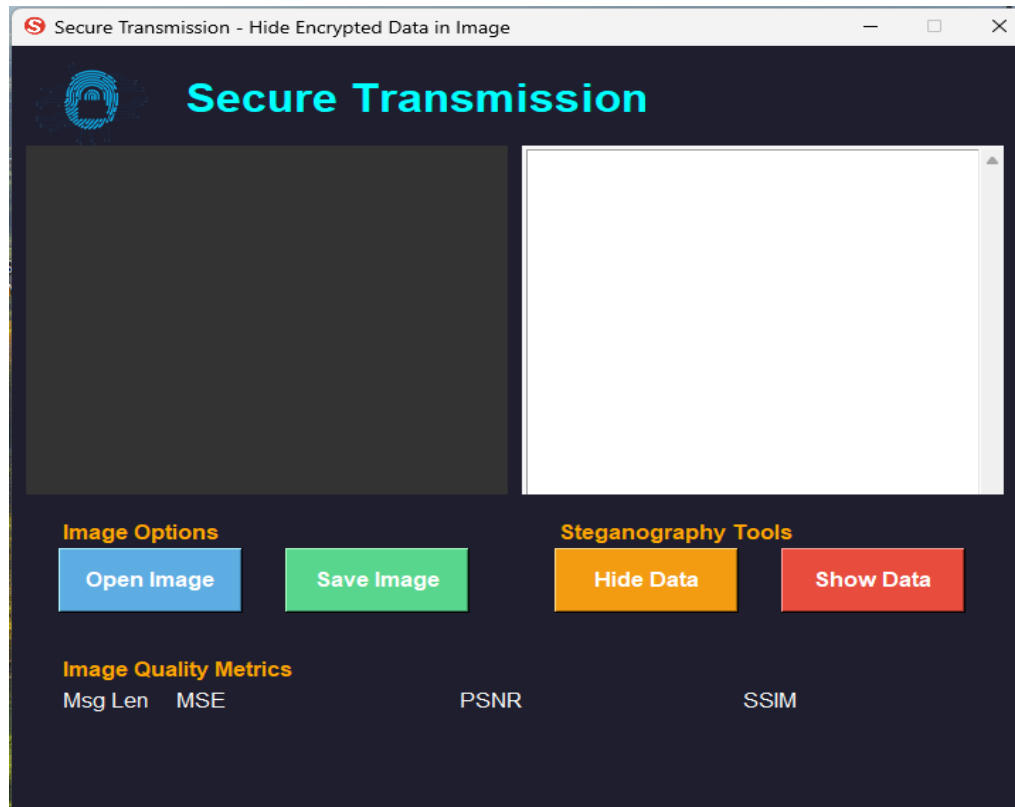


Fig.4.2.1 Main window

The Fig.4.2.1 shows a user-friendly interface of a software tool called "Secure Transmission - Hide Encrypted Data in Image." At the top, the title "Secure Transmission" stands out in bright cyan, giving it a clear and modern look. The layout includes two main sections: a dark box on the left, likely used to display the selected image, and a white box on the right, probably for entering or viewing hidden text. Just below these sections, you'll find a set of clearly labeled buttons. Under "Image Options," there are buttons to open and save images, making it easy to work with your files. In the "Steganography Tools" area, the Hide Data and Show Data buttons let users easily embed or extract hidden messages. At the bottom, there's space reserved for showing image quality metrics like message length, MSE, PSNR, and SSIM, which help evaluate the impact of data hiding on the image.

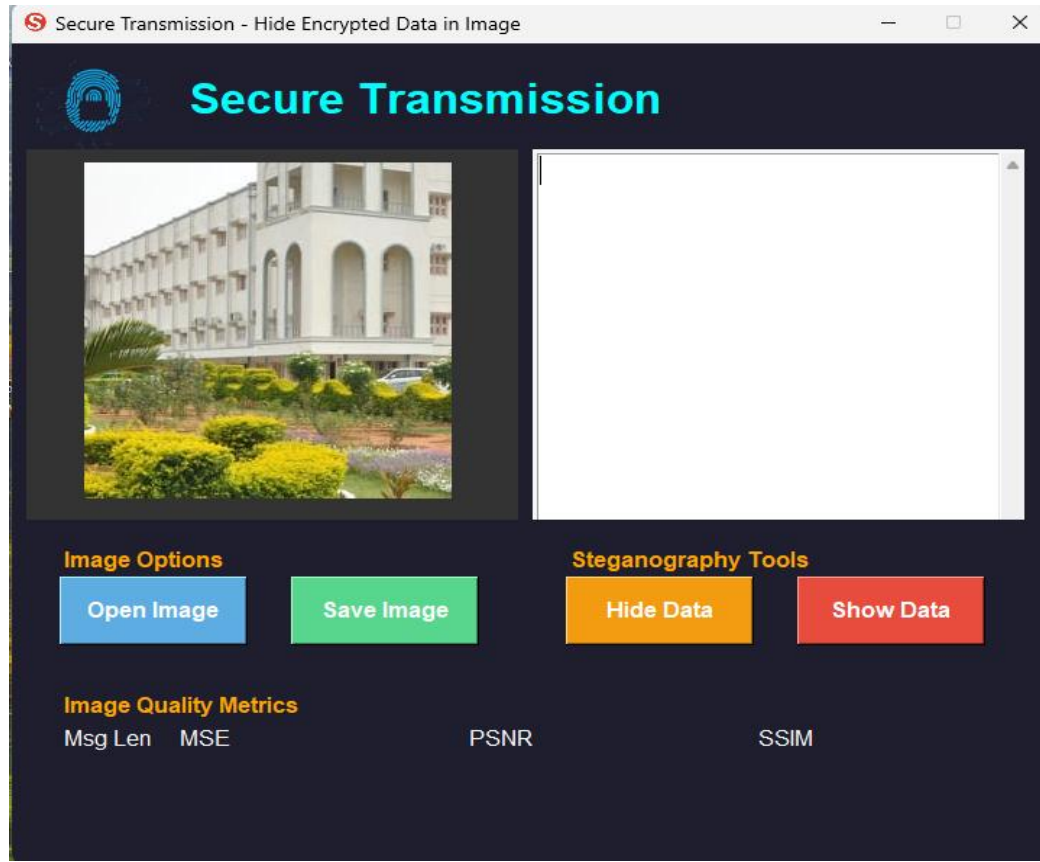


Fig.4.2.2 Shows Textbox

The Fig.4.2.2 above belongs to the software "Secure Transmission - Hide Encrypted Data in Image." The title is clearly shown in cyan at the top. On the left is a preview of a building image, while the right-side white box is for entering cipher text. Below are user-friendly buttons for opening/saving images and hiding/showing data. The bottom section displays image quality metrics like Msg Len, MSE, PSNR, and SSIM.

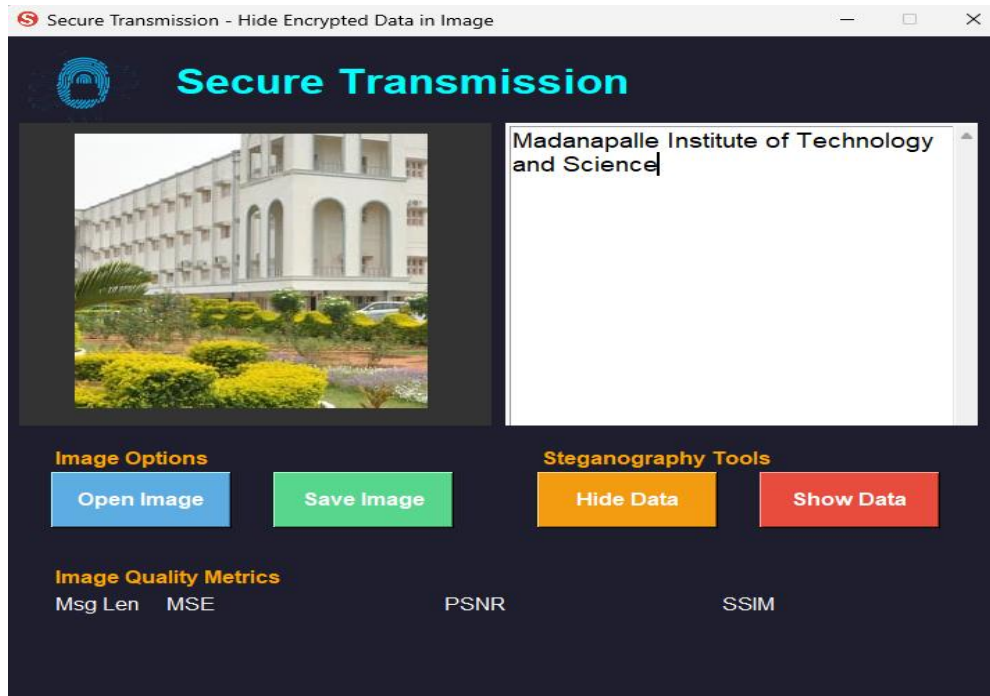


Fig.4.2.3 Enter the Secret Message

The Fig.4.2.3 above showcases the interface of a steganography tool called "Secure Transmission - Hide Encrypted Data in Image." The title is clearly highlighted at the top in bold cyan text. On the left, there's a photo of a white academic building surrounded by well-kept greenery, likely uploaded by the user. To the right, in the white input box, the text "Madanapalle Institute of Technology and Science" has been typed, which is the message intended to be hidden within the image. Just below, easy-to-use buttons let the user open or save images, and hide or reveal data. The bottom part of the screen is set to display important image quality metrics like message length, MSE, PSNR, and SSIM, once data processing is done.

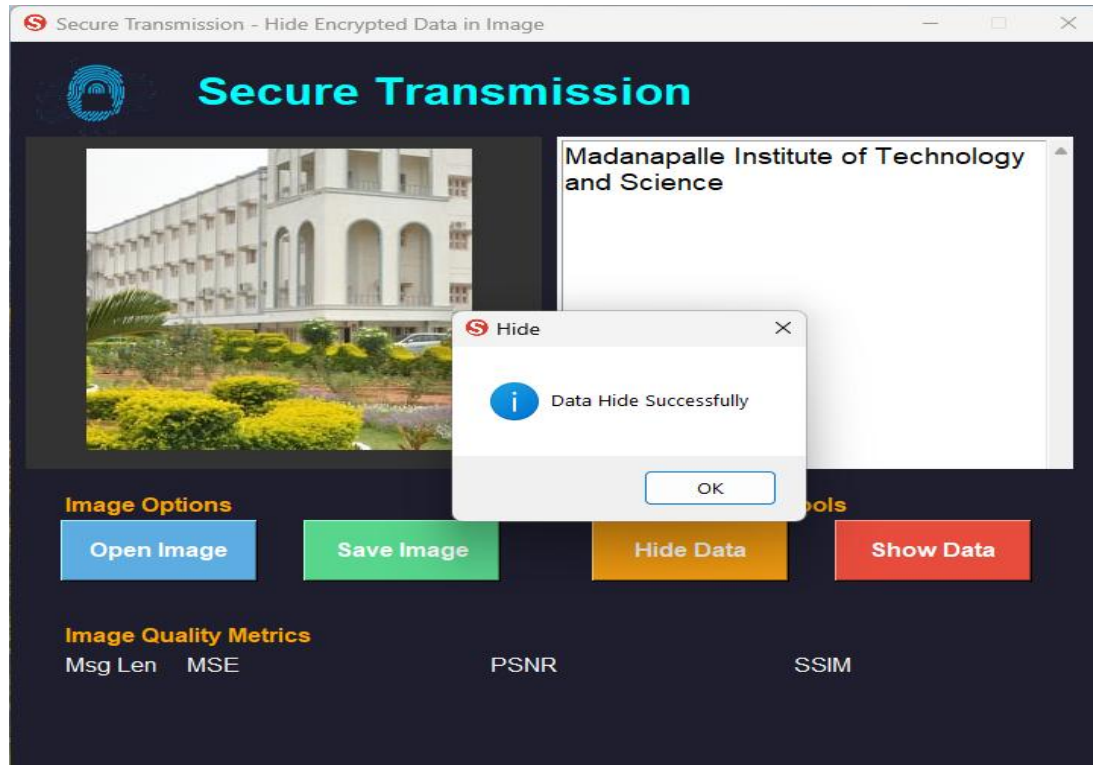


Fig.4.2.4 After Click on Hide Data Button

The Fig.4.2.4 above shows the interface of a steganography tool called "Secure Transmission - Hide Encrypted Data in Image." The title appears at the top in bright cyan, giving the interface a clean and modern look. On the left side, there's an image of a large academic building surrounded by greenery, while on the right, the user has typed the message "Madanapalle Institute of Technology and Science" into the text box. At the center of the screen, a pop-up message confirms that the data has been successfully hidden inside the image. Just below, there are simple, clearly labeled buttons for actions like opening and saving images, as well as hiding or revealing hidden data, making the tool easy to use.

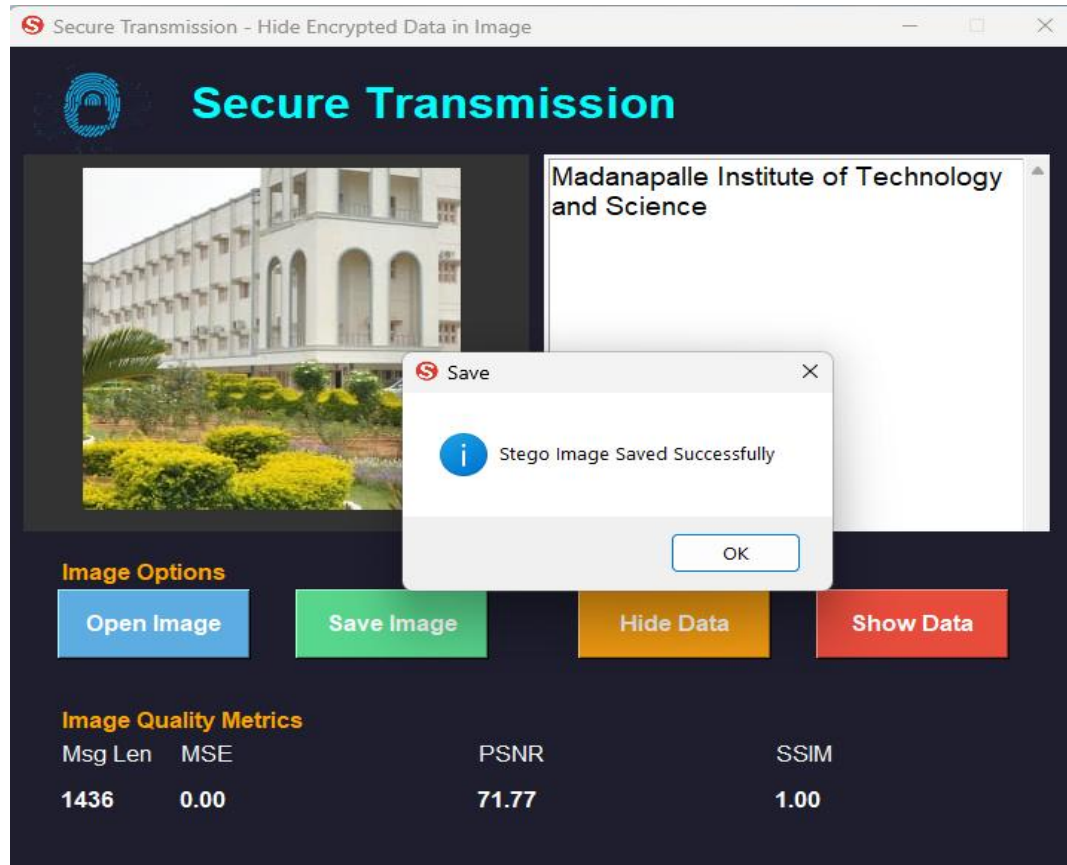


Fig.4.2.5 Popup Message of Stego- Image Saved

The Fig.4.2.5 above shows the interface of a user-friendly steganography software called "Secure Transmission - Hide Encrypted Data in Image." The title is clearly displayed at the top in a bright cyan color. On the left side, there's a picture of a large white building, and on the right, the message "Madanapalle Institute of Technology and Science" has been entered for hiding within the image. A pop-up message in the center confirms that the stego image has been saved successfully, meaning the hidden data is now embedded in the image. Just below the main section, there are buttons for actions like opening, saving, hiding, or revealing data, making the tool easy to operate. At the bottom, image quality metrics such as message length (1436), MSE (0.00), PSNR (71.77), and SSIM (1.00) are displayed, showing that the image quality remains excellent after hiding the data.

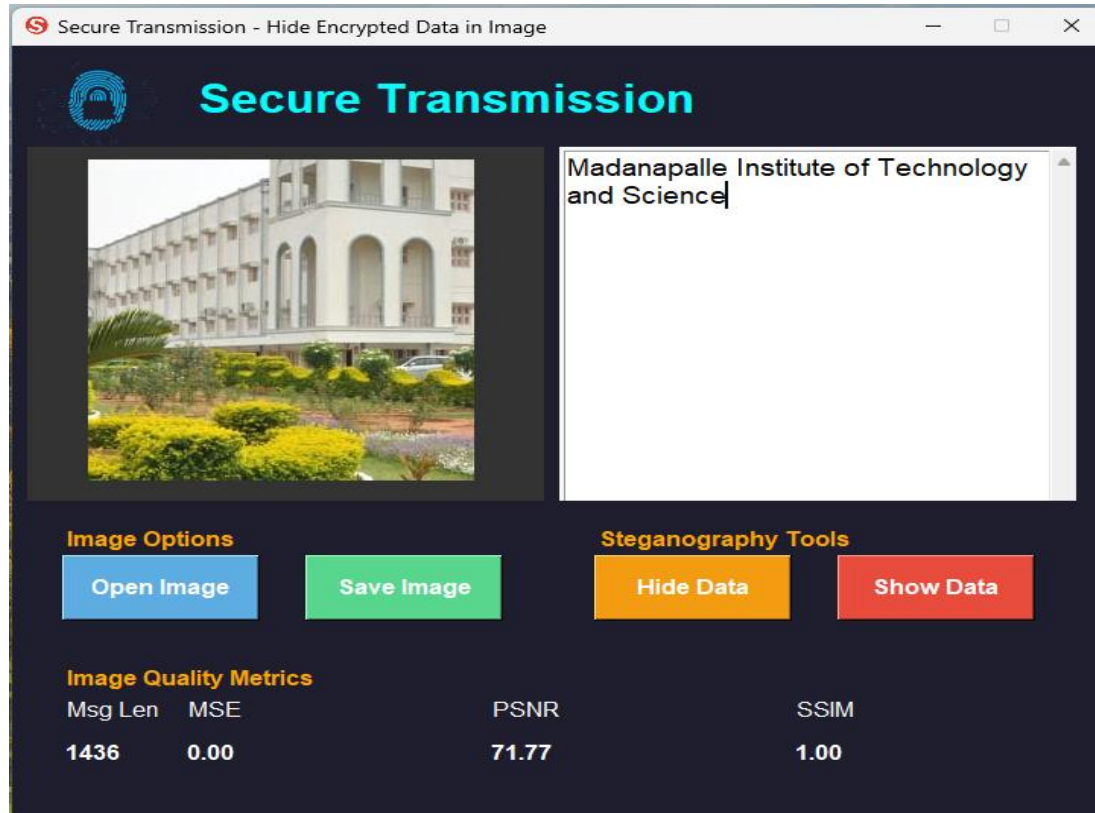


Fig.4.2.6 Results of MSE PSNR and SSIM Values

The Fig.4.2.6 above captures the final step of a successful encryption process using steganography software. A cover image has been uploaded on the left, and the message "Madanapalle Institute of Technology and Science" is now shown in the text area, confirming that it has been securely embedded into the image. The software also provides feedback on image quality after encryption—showing a message length of 1436 characters, zero distortion (MSE: 0.00), a high PSNR of 71.77, and a perfect SSIM score of 1.00. These values indicate that the visual quality of the image remains intact. This screen confirms that the sensitive data has been safely and seamlessly hidden.

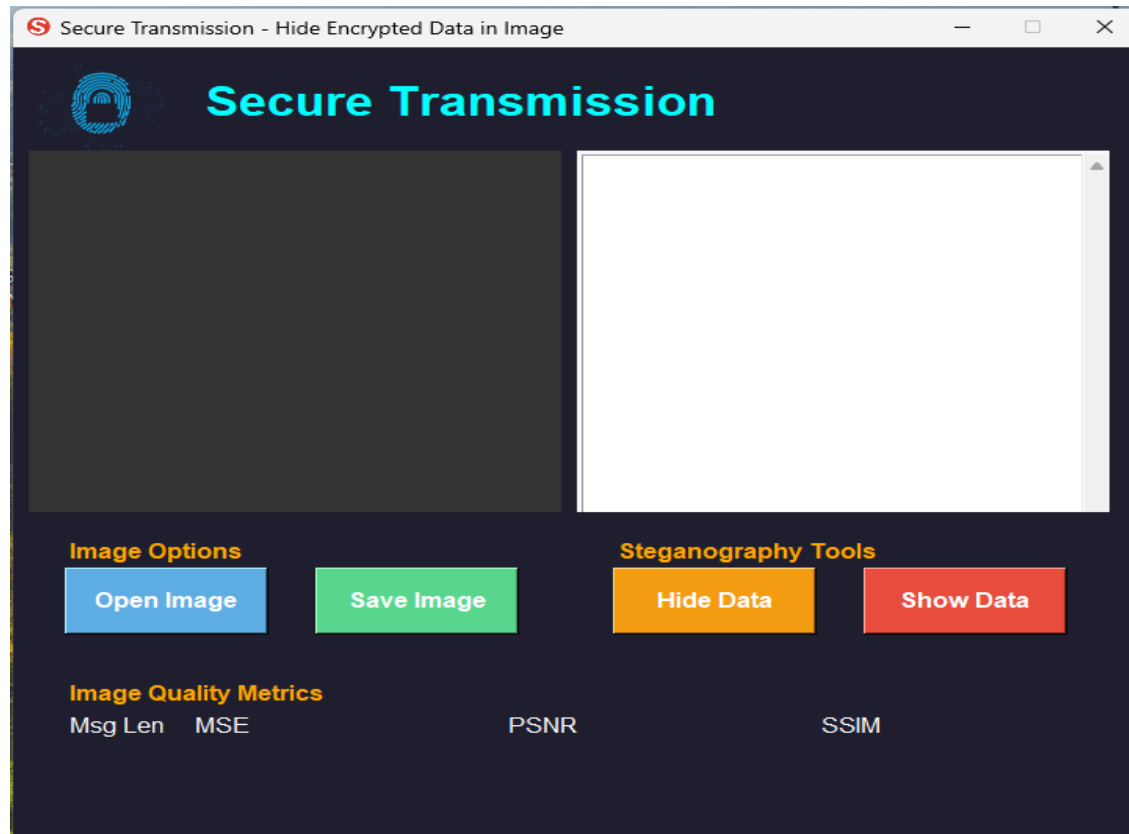


Fig.4.2.7 Selecting the Stego-Image to Retrieve Data from The Image

The Fig.4.2.7 above captures the starting screen of a steganography software titled "Secure Transmission - Hide Encrypted Data in Image." At this point, no image has been loaded yet, and both the preview area and the text box are empty. This is the stage where the user is expected to select a stego image by clicking the "Open Image" button. Once an image containing hidden data is uploaded, the user can retrieve the concealed information by clicking "Show Data." The software is all set and waiting for the user to begin the process of extracting or embedding data.

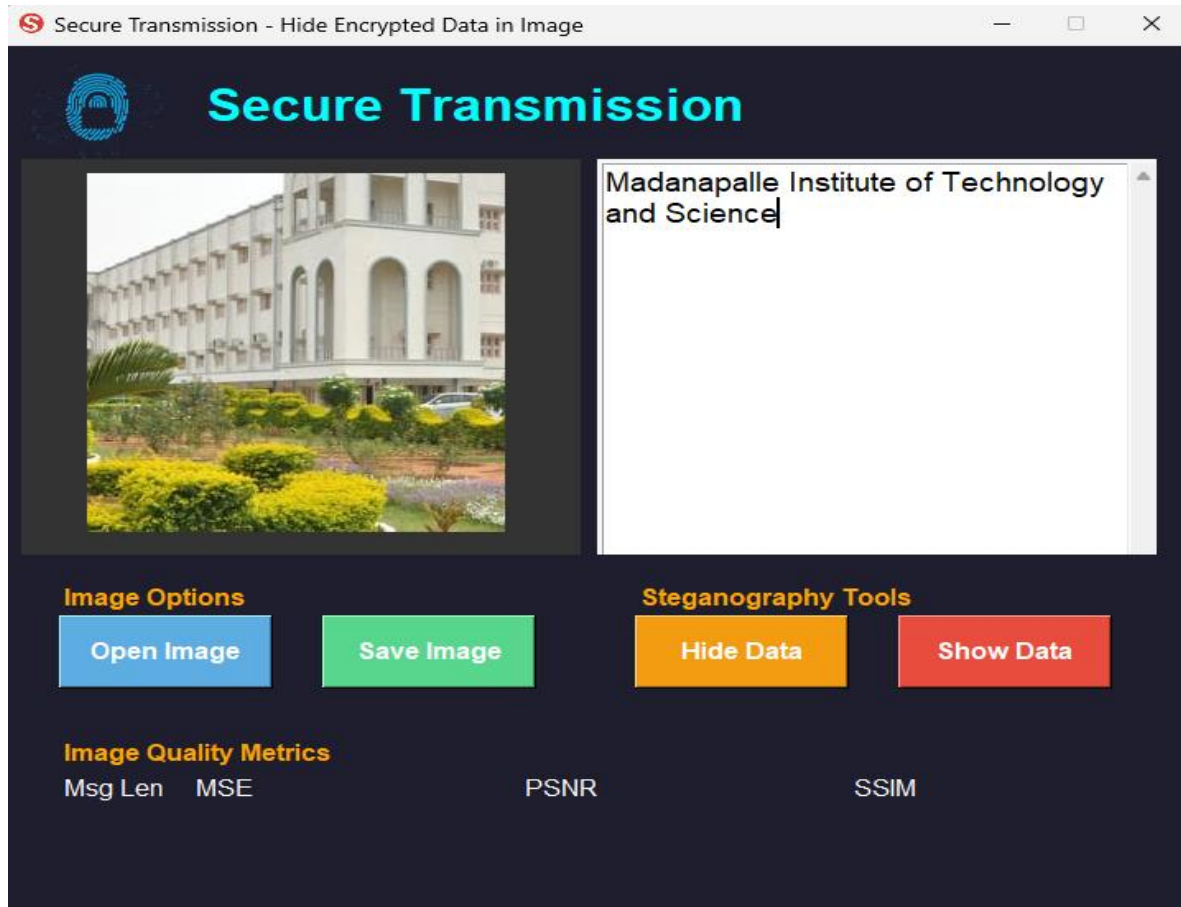


Fig.4.2.8 After click on show data button the data will be shown on screen

The Fig.4.2.8 above captures the successful outcome of a steganography process using the software "Secure Transmission - Hide Encrypted Data in Image." The title is prominently displayed at the top in cyan, giving the interface a clean and clear appearance. On the left, an image of a white institutional building has been uploaded, which serves as the stego image. On the right, the message "Madanapalle Institute of Technology and Science" appears in the text area—this text has been extracted from the uploaded stego image, not entered manually.

4.3. SAMPLE CODE

```
from flask import Flask, render_template, request, redirect, url_for, send_file
from werkzeug.utils import secure_filename
import os
from Cryptography import RSA_Encryption, RSA_Decryption
from Huffman import compress, decompress
from Stegano import encode_image, decode_image

app = Flask(__name__)
UPLOAD_FOLDER = 'uploads'
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER

if not os.path.exists(UPLOAD_FOLDER):
    os.makedirs(UPLOAD_FOLDER)

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/encrypt', methods=['GET', 'POST'])
def encrypt():
    if request.method == 'POST':
        message = request.form['message']
        image = request.files['image']
        filename = secure_filename(image.filename)
        image_path = os.path.join(app.config['UPLOAD_FOLDER'], filename)
        image.save(image_path)

        # Step 1: RSA Encryption
        encrypted_msg, key = RSA_Encryption(message)
```

```

# Step 2: Huffman Compression
compressed_msg = compress(encrypted_msg)

# Step 3: Steganography - Embed in Image
stego_path = encode_image(image_path, compressed_msg)

return send_file(stego_path, as_attachment=True)

return render_template('encrypt.html')

@app.route('/decrypt', methods=['GET', 'POST'])
def decrypt():
    if request.method == 'POST':
        stego_image = request.files['stego_image']
        key = request.form['key']
        filename = secure_filename(stego_image.filename)
        image_path = os.path.join(app.config['UPLOAD_FOLDER'], filename)
        stego_image.save(image_path)

        # Step 1: Extract message using Steganography
        compressed_msg = decode_image(image_path)

        # Step 2: Huffman Decompression
        encrypted_msg = decompress(compressed_msg)

        # Step 3: RSA Decryption
        decrypted_msg = RSA_Decryption(encrypted_msg, key)

        return render_template('decrypt.html', message=decrypted_msg)
    return render_template('decrypt.html')

```

```

if __name__ == '__main__':
    app.run(debug=True)

from rsa_python import rsa

class Cryptography:
    def __init__(self, key_size):
        self.key_size = key_size
        self.key_pair = rsa.generate_key_pair(self.key_size)
        self.public_key = self.key_pair["public"]
        self.private_key = self.key_pair["private"]

    def encrypt(self, message):
        return rsa.encrypt(message, self.public_key, self.key_pair["modulus"]),
        self.key_pair["modulus"]

    def decrypt(self, message):
        return rsa.decrypt(message, self.private_key, self.key_pair["modulus"])

    def customDecrypt(self, message, key, n):
        return rsa.decrypt(message, key, n)

    def get_public_key(self):
        return self.public_key

    def get_private_key(self):
        return self.private_key

    def get_key_size(self):
        return self.key_size

```

```

import heapq
from heapq import heappop, heappush
from Cryptography import Cryptography

def isLeaf(root):
    return root.left is None and root.right is None

class Node:
    def __init__(self, ch, freq, left=None, right=None):
        self.ch = ch
        self.freq = freq
        self.left = left
        self.right = right

    def __lt__(self, other):
        return self.freq < other.freq

def encode(root, s, huffman_code):
    if root is None:
        return
    if isLeaf(root):
        huffman_code[root.ch] = s if len(s) > 0 else '1'
    encode(root.left, s + '0', huffman_code)
    encode(root.right, s + '1', huffman_code)

global msg
msg = []

def decode(root, index, s):
    if root is None:
        return index

```



```

    if isLeaf(root):
        msg.append(root.ch)
        return index
    index = index + 1
    root = root.left if s[index] == '0' else root.right
    return decode(root, index, s)

def buildHuffmanTree(text):
    if len(text) == 0:
        return
    freq = {i: text.count(i) for i in set(text)}
    pq = [Node(k, v) for k, v in freq.items()]
    heapq.heapify(pq)
    while len(pq) != 1:
        left = heappop(pq)
        right = heappop(pq)
        total = left.freq + right.freq
        heappush(pq, Node(None, total, left, right))
    root = pq[0]
    huffmanCode = {}
    encode(root, "", huffmanCode)
    print('Huffman Codes are:', huffmanCode)
    print('The original string is:', text)
    s = ""
    for c in text:
        s += huffmanCode.get(c)
    print('The encoded string is:', s)
    return root, s

def decodeValues(root, s):
    if isLeaf(root):
        while root.freq > 0:

```

```
    msg.append(root.ch)
    root.freq = root.freq - 1
else:
    index = -1
    while index < len(s) - 1:
        index = decode(root, index, s)
return "".join(msg)
```

CHAPTER-V

SYSTEM TESTING

5.1 Testing

The main goal of testing is to identify and uncover errors. It involves evaluating a product to find any possible faults or weaknesses. Testing helps verify the correct functionality of components, sub-components, complete assemblies, or the final product. It is the act of running software to ensure that the system behaves as expected, meets all requirements and user needs, and does not fail in a problematic way. There are different types of testing, and each one is designed to fulfill a specific testing objective.

Types of Tests

Unit Testing

Unit testing focuses on verifying individual modules, which are the smallest units of software. Each module is tested using its design and process specifications to identify internal issues before integration testing begins. In this project, modules like image upload, profile editing, name search, and login are treated as separate units and tested with various inputs. The goal is to ensure each module performs accurately during development. User inputs are validated, and the developer tests the system modules thoroughly. Unit testing helps in early error detection, which might otherwise be hidden during component interaction.

Link Testing

Link testing examines how each module integrates into the system, not the software itself. The compatibility of each module is the main issue. The programmer runs tests on modules that have varying lengths, types, and other design characteristics.

Integration Testing

Integration testing is performed after unit testing to ensure that individual modules work together properly. The main focus is on verifying the interfaces and interactions between different modules. It helps evaluate how well the system design holds when components are integrated. In this project, all modules were combined to build the complete system. During integration, various input combinations were tested to check if the integration impacted the services' performance. This ensured that each service continued to function smoothly after being integrated.

System Testing

This tests the software system as a whole. The requirements document serves as the process's reference document, and the objective is to determine if the program satisfies those requirements. Here, the complete "ATM" has been tested against the project specifications to see whether or not all of the requirements have been met.

Acceptance Testing

Acceptance To show that the program is operating successfully, a test is run using actual client data. Here, the emphasis of testing is on the system's exterior behavior rather than the underlying logic of the program. The purpose of test cases is to exercise as many properties of an equivalency class as possible at once. A crucial stage in the creation of software is testing. It is the process of identifying mistakes and incomplete tasks as well as a thorough verification to ascertain whether the goals are fulfilled and the needs of the user are satisfied.

White Box Testing

Using this unit testing approach, each unit is evaluated in-depth at the statement level in order to identify the greatest number of problems that might occur. I verified each line of code one by one, making sure that each statement was run at least once. Glass Box Testing is another name for white box testing. For the purpose of verifying every conceivable combination of execution pathways through the code at each module level, I have created a list of test cases and sample data.

Black Box Testing

This testing method views a module as a whole unit, focusing on its interfaces and interactions rather than its internal code. The module is treated like a black box, where inputs are given, and outputs are observed without knowing the internal logic. The output is then used by other modules based on the input combinations provided. In networking, a tool called Ping is used to check if a specific host is reachable over an IP network. It sends ICMP packets to the target host and waits for a response. Ping helps measure response time and packet loss to evaluate network connectivity.

5.2 Test Objectives

- Every input field should function correctly and accept valid data.
- All pages should open smoothly when their respective links are clicked.
- Entry screens, system messages, and responses should appear promptly without any noticeable delays.

Features To Be Tested

- Making sure all user inputs follow the correct format.
- Preventing duplicate entries in the system.
- Checking that different parts of the system work well together.
- Testing input and output functions to confirm they behave as expected.

Types of Testing Performed

System Testing

We tested the entire system as a whole to see if it works as planned. This included checking that all features align with what was originally specified during the planning stage.

Code Testing

This testing focused on the logic inside the code. We ran different test inputs to make sure every line of code and every condition was triggered at least once. This helped confirm that all paths in the program are functioning.

Unit Testing

Each feature, like uploading images, logging in, editing profiles, and searching, was tested individually. These smaller parts, or modules, were given different inputs to ensure they worked correctly before being combined into the full system.

Link Testing

We tested how each part of the system connects and communicates with the others. The goal was to ensure compatibility between modules and make sure that they pass data correctly from one to another.

Integration Testing

After individual testing was complete, we combined all the parts and tested them together. We used various inputs to confirm that integration didn't break any functionality and that all services continued to work properly when joined.

Result: Integration testing was successful, and all services interacted correctly without errors.

System Testing (Full Validation)

We again evaluated the system as a whole, but this time with a focus on verifying that it met every requirement mentioned in the project documentation. This ensured complete alignment with the initial goals.

Result: The software passed all system-level checks and met every requirement.

Acceptance Testing

To simulate real-world use, we ran the software with actual user data. The goal was to confirm that it behaved as expected from a user's point of view. This kind of testing focuses more on results and ease of use than technical accuracy.

Result: The system met all expectations and was ready for use without any issues.

White Box Testing

This approach, also called glass-box testing, focuses on examining the internal structure and logic of the code. We used a set of sample inputs to test all possible execution paths, making sure that each part of the code was thoroughly checked for complete coverage.

Black Box Testing

In this test, we didn't look at the internal code. Instead, we tested the system from an outside perspective, focusing on inputs and outputs. We treated each part of the system as a "black box"

and verified that it produced the right results without errors when given certain inputs.

5.3 Test Cases Passed

Test No	Test Case Description	Expected Output	Actual Output	Result
1	Importing the predefined and user-defined modules	Imported	Imported without errors	Pass
2	Executing application page to view main window of application	Executed	Executed successfully and main window opened	Pass
3	Generate public and private keys for encryption and decryption	Generate keys	Keys generated successfully	Pass
4	Enter the plain text in the input field	Input taken	Input taken successfully	Pass
5	Convert plain text into cipher text using generated keys	Convert the text	Converted plain text into cipher text successfully	Pass
6	Hide compressed data into stego image using LSB	Hide the data into the image	Compressed data hidden into image successfully	Pass
7	Upload the stego image in PNG format	Upload the stego image	Uploaded the stego image successfully	Pass
8	Fetch the hidden data from the stego-image	Retrieve the data from the image	Retrieved the data from the image successfully	Pass
9	Decompress the fetched data using compression algorithm	Decompress the data	Decompressed the data successfully	Pass

5.4 Test Cases Failed

Test No	Test Case Description	Expected Output	Actual Output	Result
1	Select the images of JPEG format	Upload the images	Successfully uploaded the images, but only JPEG format is supported	Failed
2	Enter the data in the input field	Enter the data	Entered alphabetic data without special symbols and spaces	Failed
3	Convert the plain text into the cipher text using the generated public key	Convert plain text to cipher text using public key	Converted the plain text to cipher text without using public key	Failed
4	Compress the data to reduce the size of the data	Compress the data	Data is compressed successfully, but without using the proper schema	Failed
5	Upload the stego image to fetch data from the image of PNG format	Upload the stego image	Uploaded stego image, but only PNG format supported—other formats not supported	Failed

CHAPTER-VI

CONCLUSION & FUTURE ENCHANCEMENTS

6.1 Conclusion

The secure transmission of asymmetric based encrypted data using compression-based steganography is an effective method of ensuring the confidentiality, integrity, and authenticity of data transmission. This method combines the strengths of asymmetric encryption and compression-based steganography, providing end-to-end encryption and hiding the data within a compressed file to make it even harder for an attacker to intercept or tamper with the data.

The proposed work evolved strong asymmetric encryption algorithm RSA Cryptography to encrypt the data, as well as using compression-based steganography to further hide the data within a compressed file. The data is also authenticated using digital signatures, ensuring that the receiver can verify the identity of the sender and that the data has not been tampered with in transit. Additionally, hashing can be used to ensure the integrity of the data, and a secure communication channel to transmit the data.

Overall, the proposed work provides a high level of security for data transmission, ensuring that the data remains confidential, and cannot be tampered or intercepted in transit. While no system can be completely fool proof, this method provides strong protection against most attacks, and additional security measures can be implemented where necessary. With the increasing importance of data security, the use of secure transmission of asymmetric based encrypted data using compression-based steganography is becoming more and more essential in various fields such as finance, healthcare, military and government.

Limitations and Scope for Future Enhancements

- Quantum-resistant encryption is powerful but still evolving and consumes high resources.
- Blockchain integration increases complexity, leading to latency and large storage demands.
- Advanced steganography can still be detected by expert forensic analysis tools.
- Improper adaptive compression may cause hidden data distortion or retrieval issues.
- User-friendly tool development is difficult without compromising overall system security.

6.2 Future Enhancements:

The method of securely transmitting data using asymmetric encryption and compression-based steganography holds great potential for future improvements. One major enhancement is the use of quantum-resistant encryption, which can protect against future risks posed by quantum computing. Adding blockchain integration could further strengthen security by offering a decentralized and tamper-proof environment. Steganography techniques can also be improved by using smarter compression methods that make hidden data even more difficult to detect. Additionally, enhancing the overall algorithms can improve the accuracy and secrecy of data hiding. Most importantly, developing simple and user-friendly tools would make this technology more accessible for everyday users, not just technical experts. Together, these upgrades could lead to a more secure and widely usable system for protecting sensitive information.

REFERENCES

- [1] A Detailed Review Based on Secure Data Transmission Using Cryptography and Steganography. (2023). Volume 129, Pages 2291–2318. Published March 27, 2023.
- [2] Ahmed, H. A., Mahfoud, A., & Al-Sanjary, O. I. (2025). Comprehensive Review of Cryptography and Steganography Algorithms. *J. Information Systems Engineering & Management*, 10(29s), 2468–4376.
- [3] Al-Chaab, Wafaa, et al. (2023). Secure and low-complexity medical image exchange based on compressive sensing and lsb audio steganography. *Informatica*, 47.6.
- [4] Al-Chaab, Wafaa, et al. (2023). Secure and low-complexity medical image exchange based on compressive sensing and lsb audio steganography. *Informatica*, 47.6.
- [5] Brabin, Denslin, Christo Ananth, and Sriramulu Bojjagani. (2022). Blockchain based security framework for sharing digital images using reversible data hiding and encryption. *Multimedia Tools and Applications*, 81(17), 24721–24738.
- [6] Carpentieri, Bruno, et al. (2020). Compression-based steganography. *Concurrency and Computation: Practice and Experience*, 32(8): e5322.
- [7] Chandra, S., Paira, S., Alam, S. S., & Sanyal, G. (2020). A comparative survey of symmetric and asymmetric key cryptography. In *2014 International Conference on Electronics, Communication and Computational Engineering (ICECCE)*, IEEE, pp. 83–93.
- [8] Chatterjee, D., Nath, J., Dasgupta, S., & Nath, A. (2016). A new Symmetric key Cryptography Algorithm using extended MSA method: DJSA symmetric key algorithm. In *2011 International Conference on Communication Systems and Network Technologies*, IEEE, pp. 89–94.
- [9] Doz, Y. L., & Kosonen, M. (2020). Embedding strategic agility: A leadership agenda for accelerating business model renewal. *Long Range Planning*, 43(2–3), 370–382.
- [10] Elbirt, A. J., & Paar, C. (2015). An instruction-level distributed processor for symmetric-key cryptography. *IEEE Transactions on Computers*.
- [11] Elbirt, A. J., & Paar, C. (2015). An instruction-level distributed processor for symmetric-key cryptography. *IEEE Transactions on Parallel and Distributed Systems*, 16(5), 468–480.
- [12] Haverkamp, I., & Sarmah, D. K. (2024). Evaluating the Merits and Constraints of Cryptography Steganography Fusion: A Systematic Analysis. *Computers & Security*, 104, 102–

118.

- [13] Huang, Yingkai, et al. (2024). Robust image steganography against JPEG compression based on DCT residual modulation. *Signal Processing*, 219, 109431.
- [14] Jirwan, N., Singh, A., & Vijay, S. (2023). Review and analysis of cryptography techniques. *International Journal of Scientific and Engineering Research*, 4(3), 1–6.
- [15] Kanimozhi, R., & Padmavathi, V. (2025). Robust and Secure Image Steganography with Recurrent Neural Network and Fuzzy Logic Integration. *Scientific Reports*, 15:13122.
- [16] Li, X., & Zhao, Z. (2024). Robust Image Steganography via Color Conversion. *IEEE Transactions on Circuits and Systems for Video Technology*.
- [17] Nath, A., Ghosh, S., & Mallick, M. A. (2019). Symmetric Key Cryptography Using Random Key Generator. In *Security and Management*, pp. 234–242.
- [18] Ozdemir, S., & Xiao, Y. (2009). Secure data aggregation in wireless sensor networks: A comprehensive overview. *Computer Networks*, 53(12), 2022–2037.
- [19] Pandey, Binay Kumar, et al. (2022). Application of Integrated Steganography and Image Compressing Techniques for Confidential Information Transmission. *Cyber Security and Network Security*, 169–191.
- [20] Pandey, Binay Kumar, et al. (2022). Application of Integrated Steganography and Image Compressing Techniques for Confidential Information Transmission. Authors: Binay Kumar Pandey, Digvijay Pandey, Subodh Waria, Gaurav Agarwal, Pankaj Dadeech, Sanwta Ram Dogiwal, Sabyasachi Pramanik. Published March 27, 2022.
- [21] Qi, Y., Chen, K., Zhao, N., Yang, Z., & Zhang, W. (2024). Provably Secure Robust Image Steganography via Cross Modal Error Correction. *arXiv:2412.12206*.
- [22] Secure Communication Method Based on Encryption and Steganography. (2017).
- [23] Singh, Amit Kumar, et al. (2021). Joint encryption and compression-based watermarking technique for security of digital documents. *ACM Transactions on Internet Technology (TOIT)*, 21(1), 1–20.
- [24] Singh, K. K., & Dwivedi, S. (2022). Digital Watermarking using Asymmetric Key Cryptography and Spatial Domain Technique. *International Journal of Advance Research in Computer Science and Management Studies*, 2(8), 1–15.

- [25] Steganography using Improved LSB Approach and Asymmetric Cryptography. (2020). Presented at IEEE International Conference on Advent Trends in Multidisciplinary Research and Innovation (ICATMRI).
- [26] Sultana, S. A., Parah, M., Hussan, & Malik, B. (2021). Double Layer Security Using Crypto Stego Techniques: A Comprehensive Review. *Multimedia Tools and Applications*, 80, 25055–25080.
- [27] Vijayakumar, P., Azees, M., Kannan, A., & Deborah, L. J. (2021). Dual authentication and key management techniques for secure data transmission in vehicular ad hoc networks. *IEEE Transactions on Intelligent Transportation Systems*, 17(4), 1015–1028.
- [28] Wang, X., Wang, X., Zhao, J., & Zhang, Z. (2016). Chaotic encryption algorithm based on alternant of stream cipher and block cipher. *Nonlinear Dynamics*, 63(4), 587–597.
- [29] Yang, H., Xu, Y., Liu, X., & Ma, X. (2023). PRIS: Practical Robust Invertible Network for Image Steganography. *arXiv:2309.13620*.
- [30] Ye, H., Zhang, S., Jiang, S., Liao, J., Gu, S., Zheng, D., Wang, C., & Li, C. (2024). Robust Message Embedding via Attention Flow Based Steganography (RMSteg). *arXiv:2405.16414*.