

Robust Image Steganography Leveraging AES and RSA- Based Dual Encryption

project submitted in partial fulfilment of the requirements

for the award of the degree of

Bachelor of Engineering

In

ELECTRONICS AND COMMUNICATION ENGINEERING

Submitted by

Syed Akmal Hussain (160321735043)



Deccan College of Engineering and Technology

(Deccan Group of Institutions)

(Affiliated to Osmania University)

Department of Electronics and Communication Engineering

Dar-us-Salam, Hyderabad- 500001

2024-2025.

Certificate

This is to certify that the project work entitled “**Robust Image Steganography Leveraging AES and RSA-Based Dual Encryption**” being submitted by **Syed Akmal Hussain (160321735043)** student of ECE Department, **Deccan College of Engineering and Technology** Affiliated to **Osmania University** in partial fulfilment of the requirements for the award of the degree of **Bachelor of Engineering in Electronics and Communication Engineering**, the bonafide work carried out by us during the academic year **2024-2025**.

Internal Guide

Dr. A.Bhuvaneshwari
Associate Professor
ECE Department
DCET

Professor & Head of the Department

Electronics and Communication Engineering,
Deccan College of Engineering & Technology

Declaration

I declare that the work reported in the project entitled **“Robust Image Steganography Leveraging AES and RSA-Based Dual Encryption”** is a record of work done by me in the **Department of Electronics and Communication Engineering (ECE), Deccan College of Engineering and Technology.**

No part of this project is copied from books/ journals/ internet and wherever referred, the same has been duly acknowledged in the text. The reported data is based on the project work done entirely by us and not copied from any source.

Syed Akmal Hussain (160321735043)

Place: Hyderabad

Date:

ACKNOWLEDGEMENT

I express our deepest gratitude to our guide, **Dr A.Bhuvaneshwari** Associate Professor, ECE Department, Deccan College of Engineering and Technology for giving us the opportunity to work on this project. She was always there to provide insightful thinking needed for the completion of the project. She has been a constant source of inspiration and guidance. Her extensive discussions, invaluable guidance and friendly cooperation have been very helpful throughout the duration of this project.

I am extremely thankful to **Dr Syeda Gauhar Fatima**, Professor & HOD, ECE Department, at DCET for her valuable advice and suggestions.

I would also like to extend my thanks to **Dr. Mir Iqbal Faheem**, Director, Deccan Group of Institutions, Deccan College of Engineering and Technology, Dar us Salam, Hyderabad. I am thankful to the faculty of ECE Department, DCET for their support.

Finally, I offer my heartiest appreciation to our parents, family members and friends for their selfless blessing, encouragement and moral support in completing this project.

ABSTRACT

This project introduces a secure and adaptive image steganography system designed to overcome the common shortcomings of traditional data hiding techniques, such as limited security and visible image distortion. The core objective is to ensure both the confidentiality of sensitive data and its imperceptibility when embedded in digital images. To achieve this, the system utilizes a hybrid encryption model that combines the strengths of Advanced Encryption Standard (AES) for fast and secure symmetric encryption, and RSA for safe asymmetric key exchange. The data is first compressed and encrypted using AES to reduce its size and enhance security, while the AES key itself is encrypted using RSA, ensuring that unauthorized access remains highly improbable even if the stego-image is intercepted. Once encrypted, the data is embedded into a cover image using an Adaptive Least Significant Bit (LSB) Matching algorithm. This method improves upon standard LSB techniques by selectively targeting image regions that are less perceptible to the human visual system, thereby minimizing visible artifacts. Additionally, randomized pixel selection is employed to further obfuscate embedding patterns, making the hidden data more resistant to detection via statistical analysis or steganalysis tools.

To validate the system's performance, several industry-standard image quality metrics are employed, including Peak Signal-to-Noise Ratio (PSNR), Mean Squared Error (MSE), and Structural Similarity Index (SSIM). These metrics collectively demonstrate that the system maintains high visual fidelity of the stego-image while preserving the integrity of the embedded data. This integrated approach holds significant promise across sectors where secure and covert data transmission is essential. Potential applications include secure communications in defence and intelligence sectors, healthcare, and copyright watermarking for digital content protection. The project delivers a robust, scalable, and efficient solution by combining strong encryption with perceptual steganography, offering a practical tool for modern data protection in a digital world. Ultimately, this system balances security, stealth, and image quality, making it a viable approach for real-world deployment in sensitive and mission critical environments.

	Contents	Page No
Abstract		i
List of Contents		ii
List of Figures		iv
List of Tables		vi
List of Abbreviations		vii
Chapter 1	Introduction	1
	1.1 Introduction	1
	1.2 Aim of the project	1
	1.3 Motivation of the Project	2
	1.4 Research methodology	2
	1.5 Applications	2
	1.6 Organization of the Report	3
Chapter 2	Literature Survey	4
Chapter 3	Steganography Algorithms	7
	3.1 DCT method	7
	3.2 DWT method	8
	3.3 LSB method	9
	3.4 Matched LSB method	10
	3.5 AES Encryption	11
	3.6 RSA Encryption	13
	3.7 Proposed System	14
Chapter 4	Proposed AR-DEA Steganography System	19
	4.1 Overview of the System	19
	4.2 Tools and Technologies Used	20
	4.3 User Interface Design and Workflow	21
	4.4 System Workflow	23
	4.5 Working Principle	23
	4.5.1 Embedding Process	23
	4.5.1.1 Hybrid Encryption (RSA and AES)	24
	4.5.2 Extraction Process	31
	4.6 Performance Evaluation Metrics	32

	4.6.1 Peak Signal-to-Noise Ratio (PSNR)	32
	4.6.2 Mean Squared Error (MSE)	32
	4.6.3 Structural Similarity Index (SSIM)	33
	4.7 Log File and RGB Pixel Change Analysis	33
	4.8 Hardware Requirements & Software Requirements	34
	4.8.1 Hardware Requirements	35
	4.8.2 Software Requirements	36
	4.9 Python Libraries and Tools Used	39
	4.10 Supported File and Image Formats	39
	4.11 IDE and Runtime Environment	39
	4.12 Setup and Installation Instructions	40
	4.13 Software Features and Functionality Summary	40
	4.14 System Flexibility and Portability	41
Chapter 5	Results & Discussions	42
	5.1 Performance Evaluation of Proposed Method	42
	5.2 Comparative Analysis with Existing Methods	46
	5.3 Summary	53
Chapter 6	Conclusion and Future Scope	54
	References	55
	Appendix:	58

List Of Figures

S.no	Title	Page no
1	Figure 1: DCT Method	7
2	Figure 2: DWT Method	8
3	Figure 3: LSB Method	9
4	Figure 4: LSB Matching Method	10
5	Figure 5: AES Encryption Block Diagram	11
6	Figure 6: RSA Encryption/Decryption	13
7	Figure 7: The Dual Channel Encryption of the proposed system	15
8	Figure 8: Proposed System Block diagram	16
9	Figure 9: VsCode IDE	19
10	Figure 10: GUI	21
11	Figure 11: AES Logo	22
12	Figure 12: RSA Logo	26
13	Figure 13: Aes Flowchart	29
14	Figure 14: RSA Flowchart	30
15	Figure 15: Proposed Method Flowchart	30
16	Figure 16: PSNR formula	32
17	Figure 17: MSE formula	32
18	Figure 18: SSIM formula	33
19	Figure 19: log file	34
20	Figure 20: vs code logo	39
21	Figure 21: Python Logo	39
22	Figure 22: Cover Image and cover image with hidden message(1)	43
23	Figure 23: Histogram comparisons (1)	43
24	Figure 24: Cover Image and cover image with hidden message (2)	44
25	Figure 25: Histogram comparisons(2)	44
26	Figure 26: Cover Image and cover image with hidden message(3)	45
27	Figure 27: Histogram comparisons (3)	45
28	Figure 28: Image Data that was used as Cover Image	46

29	Figure 29: Bar graph comparison for PSNR evaluation	47
30	Figure 30: Bar graph comparison for MSE evaluation	48
31	Figure 31: Bar graph comparison for SSIM evaluation	49
32	Figure 32: Bar graph comparison for PSNR evaluation-1	50
33	Figure 33: Bar graph comparison for MSE evaluation-1	51
34	Figure 34: Bar graph comparison for SSIM evaluation-1	52

List of Tables

S.no	Title	Page No
1	System Workflow	23
2	Excel Report Example	34
3	Minimum Hardware Requirements	35
4	Software Hardware Requirement	35
5	Software Features and Functionality Summary	40
6	Sample test case results	42
7	Imperceptibility (PSNR) comparison	47
8	Distortion (MSE Comparison):	48
9	Perceptual Quality (SSIM Score Distribution) comparison	49
10	Evaluating Proposed method & Existing methods (PSNR)	50
11	Evaluating Proposed method & Existing methods (MSE)	51
12	Evaluating Proposed method & Existing methods (SSIM	52

List of Abbreviations

Abbreviations Full Form / Description

AES	Advanced Encryption Standard
AR-DEA	Adaptive Randomized AES-RSA Dual Encryption Algorithm
BMP	Bitmap (image file format)
CBC	Cipher Block Chaining (AES mode of operation)
CSV	Comma Separated Values
EXIF	Exchangeable Image File Format
GUI	Graphical User Interface
IO	Input/Output
IV	Initialization Vector
JPEG	Joint Photographic Experts Group (image file format)
JPG	Joint Photographic Experts Group (common file extension for JPEG)
LSB	Least Significant Bit
MSE	Mean Squared Error
N/A	Not Applicable / Not Available (used in GUI labels)
NP	NumPy (Python library alias)
OAEP	Optimal Asymmetric Encryption Padding (RSA padding scheme)
PEM	Privacy-Enhanced Mail (file format for cryptographic keys)
PIL	Python Imaging Library (often referred to as Pillow)
PNG	Portable Network Graphics (image file format)
PSNR	Peak Signal-to-Noise Ratio
RGB	Red, Green, Blue (a colour model)
RSA	Rivest-Shamir-Adleman
SSIM	Structural Similarity Index
TIFF	Tagged Image File Format
Tk	Tkinter (Python's standard GUI toolkit)

List of Abbreviations

Abbreviations Full Form / Description

WEBP	WebP (image file format)
XLSX	Office Open XML Spreadsheet (Excel file format)
ZLIB	Zlib (data compression library)

CHAPTER 1

INTRODUCTION

1.1 Introduction

In current digital era sensitive information must be securely transmitted, and it becomes a vital concern. While traditional cryptography protects the content of a message, it is more exposed to interception and cryptanalysis. Steganography addresses this concern by embedding secret data within seemingly innocuous files—most commonly, digital images. This technique ensures that the presence of hidden information is not obvious to unintended recipients. Among various image steganography methods.

To overcome these limitations, this project introduces an adaptive LSB-based steganography system that intelligently embeds data using randomized pixel selection and channel prioritization. Moreover, since steganography by itself does not ensure the secrecy of the data alone, the system integrates hybrid encryption method such as Advanced Encryption Standard (AES) that encrypts symmetrically and Rivest–Shamir–Adleman (RSA) for secure asymmetric exchange. Together, these techniques provide both imperceptibility and security, enabling robust covert communication through images.

1.2 Aim & Objectives

The main aim of this project is to design and implement a secure and intelligent image steganography system that embeds encrypted data into cover images with minimal visual distortion. To achieve this aim following objectives are fulfilled

1. Extensive study of existing image steganography techniques and cryptographic algorithms—focusing on Least Significant Bit (LSB) embedding, Advanced Encryption Standard (AES), Rivest-Shamir-Adleman (RSA), and compression methods.
2. Implementation of traditional steganography methods, Discrete Cosine Transform (DCT), Discrete Wavelet Transform (DWT), for embedding & extraction of hidden message from the cover image.
3. Implementation of Least Significant Bit (LSB) and Matched Least Significant Bit (M-LSB) for image steganography.
4. Proposed Adaptive Randomized AES-RSA Dual Encryption Algorithm (AR-DEA) for secure embedding and extraction in images.

Performance evaluation and comparative analysis of the steganography algorithms.

1.3 Motivation of the Project

The increasing frequency of cyber threats, surveillance, and data breaches has highlighted the need for secure and undetectable communication channels. While encryption protects data confidentiality, it can also raise suspicion if intercepted. Steganography offers a unique advantage by hiding the existence of the data itself. However, basic LSB methods are easily detectable and do not encrypt the embedded message. Motivated by these gaps, this project proposes a solution that combines adaptive LSB steganography with hybrid encryption and compression, making the embedded data both hidden and secure. Furthermore, this system is user-friendly, supports any file type, and includes automated metric analysis, making it appropriate for practical use in real time scenarios.

1.4 Research Methodology

Our proposed system employs a structured technical approach that integrates image processing, cryptography, and data compression. Hybrid encryption is a core component, where data is first compressed using the zlib library before being encrypted with AES in CBC mode via a randomly generated key. This AES key is then itself secured using RSA with PKCS1_OAEP padding. For data embedding, it utilizes adaptive LSB Embedding matching algorithm in which the encrypted payload is hidden in the cover image. To ensure unpredictability, pixel positions are shuffled using a fixed random seed, and a consistent channel priority is applied as Blue → Green → Red. The entire system is implemented with a Tkinter-based Graphical User Interface (GUI), allowing users to easily select images and files, perform both embedding and extraction, and view important performance metrics. The development relies on Python 3.10 and key libraries including PIL for image processing, PyCryptodome for AES and RSA, Tkinter for the GUI, NumPy for numerical operations, zlib for compression, and pandas/openpyxl for data handling. Visual Studio Code serves as our Integrated Development Environment (IDE). This structured methodology ensures our final application is secure, adaptive, user-friendly, and efficient for real-time scenarios.

1.5 Applications

This hybrid steganography system offers practical utility across various sectors where secure and discreet data transmission is paramount. In military and defense, it can be used for transmitting classified orders or documents embedded discreetly within images. For healthcare, it provides a method to hide sensitive patient records or reports directly within diagnostic images, enhancing privacy. In digital forensics, the system can embed evidence or logs within image metadata or visuals, aiding investigations. It's also valuable for intellectual property protection through watermarking images with encrypted identifiers. Furthermore, it supports

corporate and financial communication by enabling the secure transmission of confidential business files.

1.6 Organization of the Report

This report is systematically organized into six distinct chapters. Chapter 1 introduces the project by outlining the problem it addresses, the motivations behind it, its core aims, the methodology employed, and its diverse applications. Chapter 2 provides a detailed literature survey, examining existing steganography and encryption techniques, discussing their advantages and limitations, and pinpointing key research gaps. Chapter 3 delves into the system design, presenting block diagrams for both existing and proposed systems, alongside an analysis of their respective pros and cons. Chapter 4 discusses the system's operational flow using flowcharts, describes the Graphical User Interface (GUI), lists the hardware and software requirements necessary for building and running the application, complete with descriptions of all utilized tools and technologies. Chapter 5 presents the results, including key performance metrics like PSNR, MSE, and SSIM. Finally, Chapter 6 offers a comprehensive conclusion and outlines the future scope for extending the project.

CHAPTER 2

LITERATURE SURVEY

This chapter delves into the existing research, helping our team understand the evolution of image steganography, hybrid cryptography, data compression, and quality evaluation metrics. This survey is crucial for identifying the gaps that our current project aims to address. For each cited reference, we've adhered strictly to the project guidelines, explaining each in 3–4 lines.

This chapter surveys existing research to help our team understand the evolution of image steganography, hybrid cryptography, data compression, and quality evaluation metrics. This review is essential for identifying the research gaps that our current project aims to address. For each cited reference, we've adhered to project guidelines, explaining each in 3–4 lines.

The field of image steganography is constantly advancing and it demands more secure and harder to detect data hiding methods. Early methods like Least Significant Bit (LSB) embedding, though simple, proved vulnerable. Fridrich et al. (2001) demonstrated statistical detection methods for LSB embedding, highlighting the need for more sophisticated strategies. To counter this, Mielikainen (2006) introduced LSB Matching, which subtly modifies pixel values (incrementing or decrementing by one) instead of direct replacement, significantly reducing statistical detectability. Further enhancing imperceptibility, Kekre et al. (2010) suggested adaptive embedding, focusing on edge regions of an image where human vision is less sensitive to changes. For a comprehensive overview, Cheddad et al. (2010) surveyed steganographic techniques, classifying them into spatial and transform domains, including methods like Discrete Wavelet Transform (DWT) and Discrete Cosine Transform (DCT). Building on these foundations, Walia et al. (2016) advanced LSB techniques by incorporating AES encryption and adaptive embedding, forming a crucial basis for many hybrid steganographic systems.

To ensure data confidentiality and integrity, cryptographic techniques are often integrated with steganography, leading to hybrid encryption models. At the forefront of symmetric key encryption, Daemen and Rijmen (2002) developed the Advanced Encryption Standard (AES), commonly recognized for its speed, strong security and efficient performance. Rivest, Shamir, and Adleman in (1978) developed RSA which is an asymmetric encryption technique and plays a major role in public key cryptography and secure key distribution. Recognizing the strengths of both, Chakrabarti and Ghosal (2014) demonstrated that combining AES and RSA provides strong data confidentiality with secure key management. Wang et al. (2012) extended this hybrid model by integrating the Blowfish algorithm to improve adaptability. Ahmed (2014)

further verified that using separate encryption for the data payload and the encryption key enhances the overall security model's robustness, even in compromised environments.

The efficiency of steganographic systems is heavily influenced by the size of the data being hidden, making data compression a vital component. Kim and Kim (2006) emphasized compression's role in reducing the embedded payload size to maintain image quality. Using libraries like zlib, which implements the DEFLATE algorithm, provides lossless compression, improving steganographic systems' effectiveness. Zaidan et al. (2010) explored integrating cryptographic and compression techniques to enhance hiding capacity. Munir and Javed (2011) combined compression with region-based embedding to improve efficiency. Noda et al. (2005) proposed measuring image complexity to determine optimal embedding regions, where compression plays a vital role.

The evaluation of steganographic systems relies on metrics quantifying imperceptibility and quality of the image. Wang et al. (2004) introduced the Structural Similarity Index (SSIM), which determine the visual quality of the image in terms of luminance, contrast, and structural consistency, offering a perceptual measure closer to human vision. Mishra and Yadav (2020) used PSNR and MSE to compare algorithms, concluding that PSNR values above 40 dB indicate high visual fidelity. Barni et al. (2001) applied pixel-wise masking to optimize stego quality without significant distortion. Zhang and Wang (2004) proposed reversible watermarking, demonstrating how distortion metrics like MSE can guide imperceptible embedding. Kharrazi et al. (2004) supported SSIM for more human-perceptual evaluations compared to PSNR alone.

The field continues to evolve with emerging trends leveraging advanced technologies. Li et al. (2020) used Generative Adversarial Networks (GANs) to develop highly undetectable steganographic models, marking a shift toward AI-based embedding. Rehman et al. (2020) integrated blockchain with steganography for tamper-proof communication and metadata tracking. Sharma and Kalra (2016) developed a hybrid scheme using DWT and RSA, showing robustness under image processing attacks. Liu et al. (2021) applied deep learning to automate embedding and decoding. Halder et al. (2019) emphasized the importance of using both encryption and steganography in tandem for layered security.

From the literature we've reviewed, it's evident that while many techniques focus on either data hiding or encryption independently, a significant research gap exists in systems that seamlessly combine both adaptive steganography and hybrid cryptography within a unified and user-friendly application. Most traditional methods often lack sufficient randomness, making them easily detectable. Our project aims to bridge this critical gap by proposing a robust and practical

system. Specifically, we're combining AES and RSA encryption with zlib compression and adaptive LSB Matching that utilizes randomized pixel selection and color channel prioritization to minimize detectability. Furthermore, our system will provide automated performance metrics and detailed logging capabilities, offering transparent insights into its operations. By merging these advanced techniques into a cohesive and user-friendly application, our system aims to be a secure, robust, and practical solution highly suitable for real-world secure communication, addressing existing limitations and offering a comprehensive approach to data protection.

CHAPTER 3

STEGANOGRAPHY ALGORITHMS

This chapter discusses the architecture and functioning of both the existing and the proposed systems. It compares their capabilities based on security, performance and practicality. The Proposed Adaptive Randomized AES-RSA Dual Encryption Algorithm (AR-DEA) is designed to overcome the limitations of basic LSB-based steganography by introducing encryption, compression, adaptivity, and randomized embedding logic.

3.1 DCT method

Discrete Cosine Transform (DCT) is a frequency domain steganographic technique that hides information in the transform domain coefficients instead of altering the pixel level. This method provides better imperceptibility and robustness, especially for lossy image formats like JPEG (Petitcolas et al., 1999).

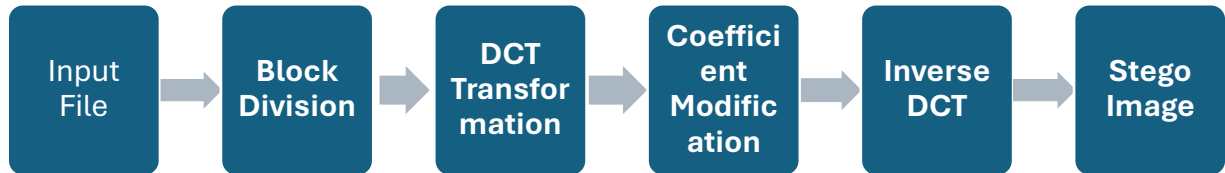


Figure 1: DCT-Based Steganography

Steps:

- **Input File:** Convert the secret file into binary data.
- **Block Division:** The image in which data is to be hidden is divided into 8×8-pixel blocks.
- **DCT Transformation:** Apply DCT to each block to transform into frequency coefficients.
- **Coefficient Modification:** Embed bits into selected mid-frequency DCT coefficients to maintain imperceptibility.
- **Inverse DCT:** Apply inverse DCT to reconstruct the modified image blocks.
- **Stego Image:** Save the modified image.

Technologies Used in Existing Systems:

- Image Format: Primarily JPEG supports DCT natively
- Compression Aware: Compatible with lossy formats
- Sometimes includes encryption or password
- Limited GUI use; mostly academic tools

Advantages:

- High robustness against compression and cropping.
- Embedding in frequency domain reduces visual artifacts.

- More secure than spatial domain techniques.

Disadvantages:

- Lower embedding capacity than LSB.
- Slower processing due to DCT and inverse DCT.
- Complexity in selecting coefficients for minimal distortion.

3.2 DWT method

Discrete Wavelet Transform (DWT) is a multi-resolution frequency domain method that represents the image at various levels of detail. DWT-based steganography embeds data in wavelet coefficients, usually in sub-bands like HL, LH, or HH (Xuan et al., 2002).



Figure 2: DWT-Based Steganography

Steps:

- **Input File:** Convert the secret message/file into binary.
- **Wavelet Decomposition:** Apply single-level or multi-level DWT on the cover image.
- **Sub-Band Selection:** Choose appropriate sub-bands (like HL or HH) for embedding.
- **Coefficient Modification:** Embed binary data into selected coefficients.
- **Inverse DWT:** The embedded image is recovered by applying Inverse Discrete Wavelet Transform (IDWT).
- **Stego Image:** Save the modified image.

Technologies Used in Existing Systems:

- Image Format: BMP or PNG (lossless)
- Often includes encryption or compression
- Rarely has a user-friendly GUI

Advantages:

- Good imperceptibility and robustness.
- Resistant to compression and common image attacks.
- Supports multi-level hiding for large data.

Disadvantages:

- High computational complexity.
- May result in image quality degradation if not tuned.
- More challenging to implement and optimize.

3.3 LSB method

Traditional image steganography systems embed secret data by replacing the Least Significant Bits of the cover image using LSB techniques (Mielikainen, 2006)., This method is favored for its simplicity and large data hiding capacity. However, it is also highly exposed to quantitative steganalysis techniques (Fridrich et al., 2001) and visual inspection, especially when embedding is performed sequentially without randomness.

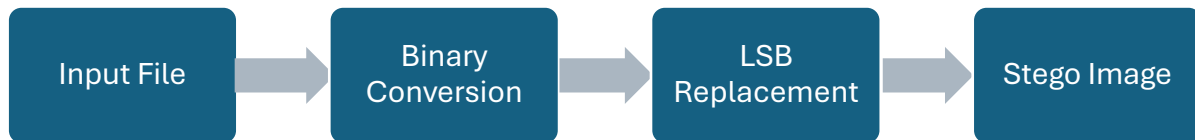


Figure 3: Traditional LSB Method

Steps:

Input File: Convert the input file to binary.

Binary Conversion: Load a cover image, convert byte data to bit data.

LSB Replacement: In this the LSBs of the pixel values in the cover image are sequentially replace the binary data.

Stego File: Save the modified image as the stego image.

Technologies Used in Existing Systems:

- Image Format: BMP or PNG
- No encryption
- No compression
- No GUI; command-line interface or scripts
- No metrics or logging

Advantages

- Simple to implement and understand.
- Fast processing due to lack of encryption/compression.
- Minimal computation resource required.

Disadvantages

- No encryption — embedded data is in plaintext (Provos & Honeyman, 2003)
- Sequential embedding — easily detectable using RS analysis
- Uniform channel usage increases distortion
- Poor robustness against image compression or modification

3.4 Matched LSB method

The Matched Least Significant Bit (LSB) method is an enhancement over the traditional LSB substitution technique. In the matched LSB method rather than embedding the secret bit in the pixels LSB, it initially compares the secret bit with pixels LSB and modifies the value if they differ. This minimizes unnecessary changes in the image, reducing distortion and improving imperceptibility(Mielikainen,2006).



Figure 4: Matched LSB Embedding

Steps:

Input File: Convert the secret file to binary form.

Bit Conversion: Load the cover image and extract its pixel values.

LSB Matching: For each bit in the secret data:

Compare with the current pixel's LSB.

If already matching, leave pixel unchanged.

If not matching, adjust the pixel value by increasing or decreasing by aligning with the target bit.

Stego Image: Save the modified image.

Technologies Used in Existing Systems:

- Image Format: BMP or PNG
- No encryption in basic versions
- No compression
- Primarily implemented via scripts or basic interfaces
- No visual adaptivity or HVS awareness

Advantages:

- Reduces number of modified pixels, improving image quality.
- Less distortion compared to direct LSB replacement.
- Slightly better resistance to statistical attacks.
- Simple and fast to implement.

Disadvantages:

- Still sequential — vulnerable to steganalysis like RS analysis.
- Embeds without considering perceptual significance (no adaptivity).

- No encryption — data remains in plaintext.
- Still not robust against compression or format conversion.

3.5 AES Encryption

The Advanced Encryption Standard (AES) developed by (Daemen and Rijmen, 2002) is a popular symmetric that works on blocks of data with similar size. A unique secret key is used for both encryption and decryption. It is known for its high security, efficiency, and extensively used for protecting critical data in different fields. AES replaced the Data Encryption Standard (DES) as the standard for government and commercial organizations globally.

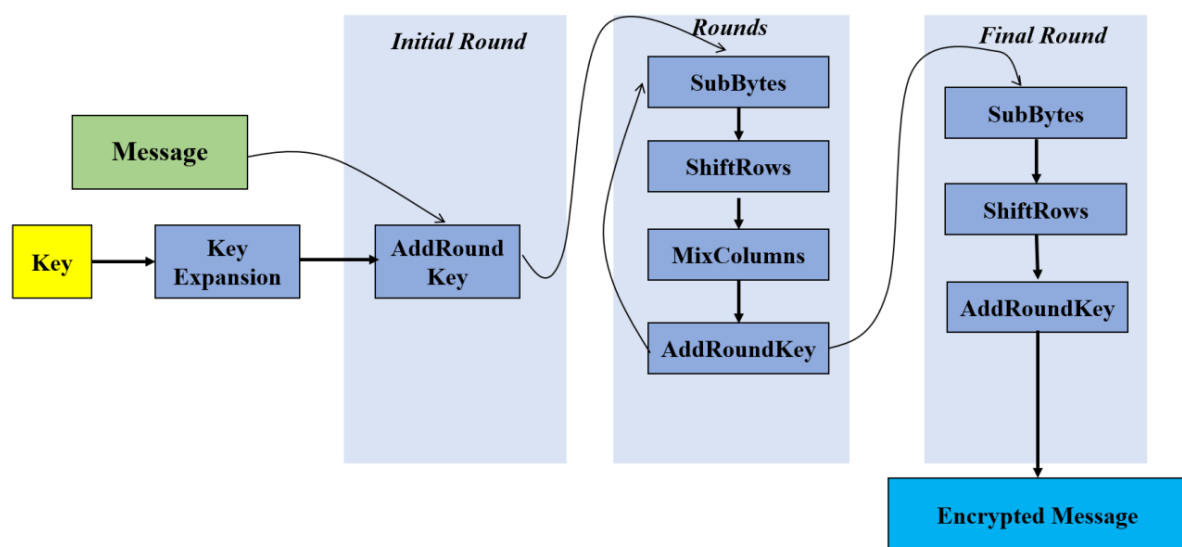


Figure 5: AES Encryption Block Diagram

Steps:

Key Expansion: In this process a sequence of round keys are derived from the main secret key and each is utilized in corresponding encryption round.

Sub Bytes: In this sub bytes are generated using a fixed substitution box (S BOX) in which each byte in the state is transformed to a different byte.

Shift Rows: Rows of the state undergoes a circular shift.

Mix Columns: Columns of the state are mathematically combined.

Add Round Key: The current round key is XORed with the state.

Initial Round: The message (data state) is merged with the first derived round key.

Rounds: Every transformation round comprises of Shift Rows, Sub Bytes, Add Round Key and Mix Columns.

Final Round: After several rounds the final round includes of Shift Rows, Sub Bytes, and Add Round Key and Mix Columns is omitted.

Encrypted Message Output: The ciphertext is output.

Technologies Used in AES-based Systems:

- Often used with modes like Cipher Block Chaining(CBC), Galois/Counter Mode (GCM) for privacy and secrecy.
- Typically relies on asymmetric cryptography (e.g., RSA or Diffie-Hellman) for secure key distribution.
- Often combined with Message Authentication Codes (MACs) or Hash-based Message Authentication Codes (HMACs).
- Integrated into network protocols like TLS/SSL, VPNs, and secure file systems.

Advantages:

- Highly secure against known attacks when implemented correctly (Daemen and Rijmen, 2002).
- It can process a large amount of data in a fast and efficient manner making it suitable for large volumes of data encryption.
- Standardized and widely supported, with hardware acceleration common in modern processors.
- Requires minimal computational resources for its level of security compared to asymmetric ciphers.

Disadvantages:

- Requires a protective and reliable approach for exchanging the secret key safely between sender and receiver, which is a significant challenge in large networks.
- As it uses a shared secret key, it cannot inherently prove who encrypted a message (sender or receiver).
- Implementations can sometimes be vulnerable to attacks that analyze power consumption or timing (Kocher et al., 1999).
- Does not inherently provide data integrity or authentication without additional mechanisms.

3.6 RSA Encryption

Rivest–Shamir–Adleman (RSA) is a foundational asymmetric public-key cryptosystem developed by (Rivest, Shamir, and Adleman, 1978). Which uses pair of public key and private key. These keys are logically and statistically related that can be shared freely for the process encryption and signature verification. Decryption and digital signing done by a secret private key. RSA is more secure as it is computationally challenging and involves the complexity of recomposing large composite numbers into the prime factors.

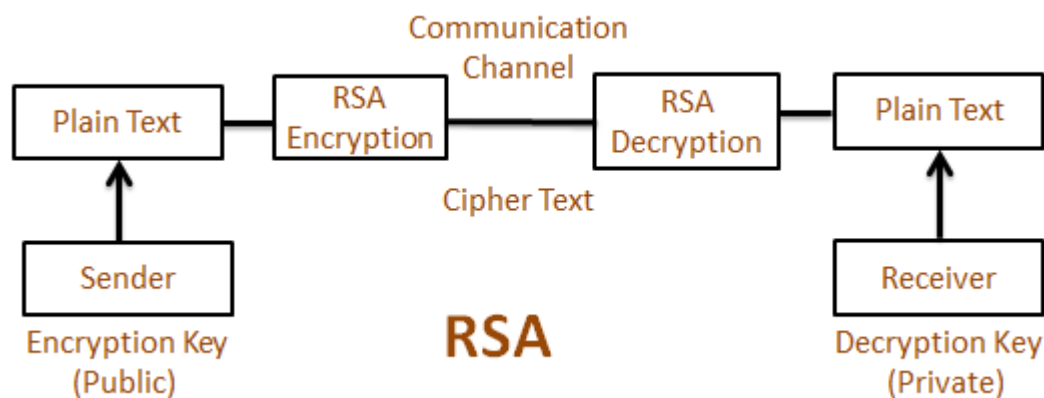


Figure 6: RSA Encryption/Decryption and Digital Signature Process

Steps:

Key Pair Generation: A pair of keys public key and private key are mathematically constructed. The public key is openly distributed, while the private key is securely hidden.

Encryption (Confidentiality): The party that sends the message encrypts it with the receiver's public key. Only the recipient's private key can decrypt the hidden message.

Decryption (Confidentiality): The receiver uses the private key to get back the original message from the ciphertext.

Digital Signature (Authentication/Integrity/Non-repudiation): The party that sends the message uses their private key to create a digital signature and appends this signature to the message. when recipient gets the message, public key from sender is used to verify the signature, which helps in authenticating and verifying integrity of the message, and preventing the sender from denying transmission.

Technologies Used in RSA-based Systems:

Public Key Infrastructure (PKI): This infrastructure is used for managing and distributing public keys via digital certificates (e.g., X.509 certificates).

Hashing Algorithms: Used to create message digests (hashes) before signing, improving efficiency (e.g., SHA-256).

Padding Schemes: Essential for security to avoid various attacks. Methods such as PKCS#1 v1.5 padding, OAEP padding are used to provide a secure key exchange and are used for digital signatures in systems like TLS/SSL, PGP, and S/MIME.

Advantages:

- **Secure Key Exchange:** Solves the issue of distribution of keys that was present in symmetric key systems; no prior shared secret is needed.
- **Digital Signatures:** Provides authentication, integrity, and non-repudiation, Digital signatures help in Verifying the sender's identity and checking the message wasn't tampered with.
- Allows secure communication over insecure channels without prior contact.
- Forms the backbone of trust in many internet security protocols (e.g., HTTPS).

Disadvantages:

- **Significantly Slower:** it has lower processing speed compared to AES and not suitable for handling large scale data encryption.
- **Key Length:** It Requires 2048-bit or 4096-bit key length to provide same security to smaller AES keys, thereby increasing the computational load.
- **Computational Intensity:** Key generation and operations are computationally intensive compared to symmetric encryption.
- **Reliance on PKI:** The security model depends on the trustworthiness of Certificate Authorities (CAs) in a Public Key Infrastructure.

3.7 Proposed System

To address the shortcoming of the current system, Proposed Adaptive Randomized AES-RSA Dual Encryption Algorithm (AR-DEA) integrates hybrid encryption (AES + RSA), adaptive LSB Matching, random pixel shuffling, and channel prioritization to enhance confidentiality, robustness, and imperceptibility. AES is used for fast symmetric encryption, while RSA secures the AES key via asymmetric encryption (Daemen & Rijmen, 2002; Rivest et al., 1978). The system also uses zlib compression to reduce payload size, thereby improving embedding efficiency and minimizing image distortion (Kim & Kim, 2006).

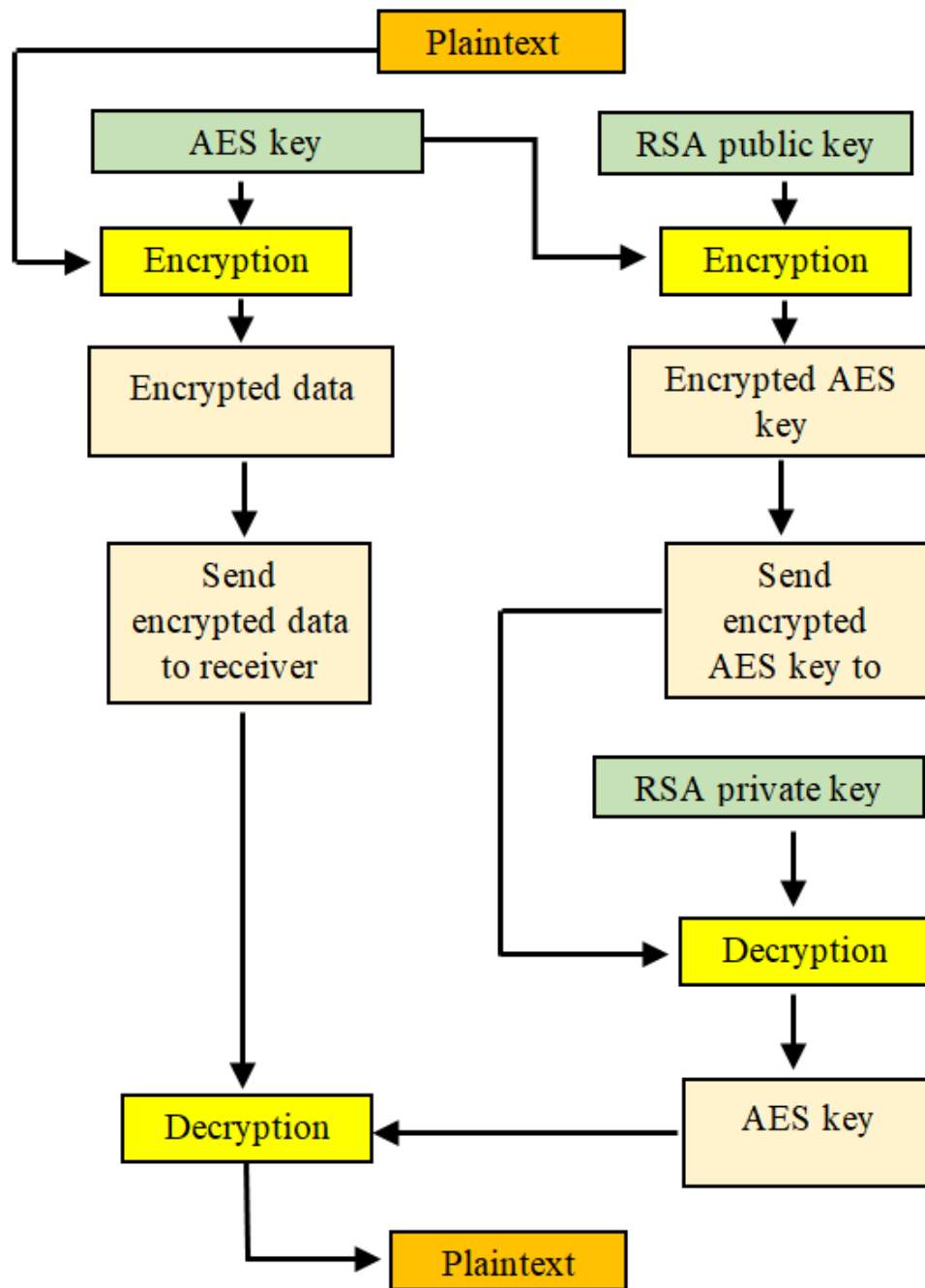


Figure 7: The Dual Channel Encryption of the proposed system

The above diagram shows how a combination of AES and RSA is used to send data securely. First, the original message (plaintext) is encrypted using a randomly generated AES key. This makes the data unreadable to others. Then, to safely send the AES key itself, it is encrypted using the receiver's RSA public key. The sender sends both the encrypted data and the encrypted AES key to the receiver. The receiver uses their RSA private key to decrypt the AES key, and then uses that AES key to decrypt the data and get back the original message. This way, the message stays secure during transmission.

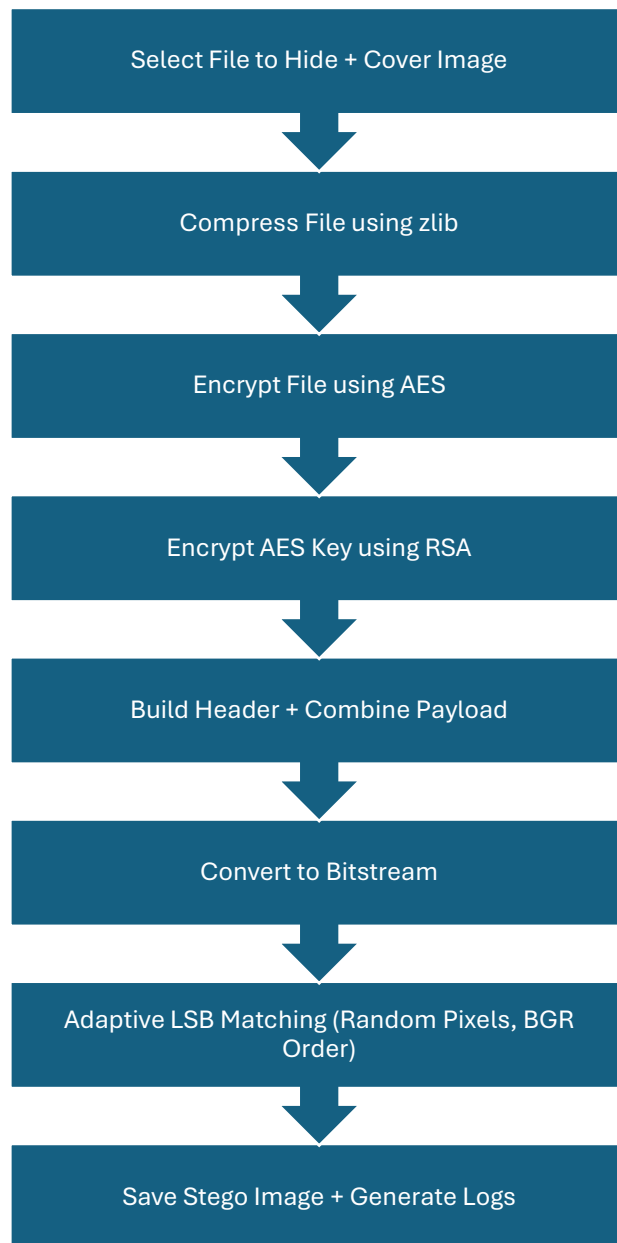


Figure 8: Proposed System

Steps:

- **Select File to Hide + Cover Image:** The user begins by selecting any file they want to hide, along with a cover image that will carry the hidden data. This cover image can be of various formats like PNG, JPG, BMP, etc.
- **Compress File using zlib technique:** The selected file is compressed using the zlib compression library. This reduces the overall size of the file, helping to improve embedding efficiency and minimize distortion in the image.

- **Encrypt File using AES:** After compression, AES encryption is applied to the compressed file. This ensures that the actual content of the file remains secure and unreadable without the correct decryption key.
- **Encrypt AES Key using RSA:** The AES key used for encryption is then itself encrypted using RSA encryption. This protects the key during transmission, as only the intended recipient with the correct RSA private key can decrypt it.
- **Build Header + Combine Payload:** A header is then created which includes metadata such as the sizes of the encrypted AES key and encrypted data. This header is combined with the encrypted payload to prepare for embedding.
- **Convert to Bitstream:** The complete payload, including the header and encrypted data, is then converted into a binary bitstream. This bitstream represents the data in 0s and 1s, ready for hiding inside the image.
- **Adaptive LSB Matching (Random Pixels, BGR Order):** Pixels of the cover image are shuffled using a fixed seed (e.g., 42) to ensure randomness. Bits from the bitstream are embedded using Adaptive LSB Matching technique, which reduces noticeable changes. Embedding follows the Blue → Green → Red channel priority to minimize visible distortion. ± 1 LSB Matching is used for making subtle pixel value adjustments when needed.
- **Save Image with hidden data + Generate Logs:** Once embedding is complete, the modified image (final image) is saved. Additionally, quality metrics like PSNR, MSE, and the total number of embedded bits are calculated and saved in log files for verification.
- **Technologies and Libraries Used:**
- **Programming Language:** Python 3.10 is used for implementing the entire application logic, GUI, and processing.
- **Libraries:**
 - Tkinter is used for building a simple graphical user interface (GUI).
 - PIL / Pillow handles image loading and saving.
 - NumPy is used for manipulating pixel arrays efficiently.
 - PyCryptodome enables AES and RSA encryption for secure file handling.
 - zlib is responsible for compressing the data before encryption.
 - pandas / openpyxl are used to log RGB differences and save the data to Excel files.

- **Supported Formats:**
- **Image for cover:** Supports a wide range of image formats including PNG, JPG, BMP, TIFF, and WebP.
- **File to be hidden:** Supports embedding of any file type, such as PDF, TXT, DOCX, ZIP, EXE, and more.
- **Advantages of the Proposed System:**
- **High Security:** AES and RSA together ensure that both the file content and the encryption key are protected, providing end-to-end encryption.
- **Data Compression:** zlib compression helps reduce the file size, which increases the capacity of the image and makes the hidden data less noticeable.
- **Adaptive Embedding:** The use of Blue-Green-Red channel order helps to embed data in a way that causes the least visible change to the image.
- **Randomization:** Shuffling pixel order with a fixed seed makes it difficult for attackers to detect patterns or recover the hidden data through statistical methods.
- **Universal File Support:** The system supports hiding any kind of file, not limited to simple text, making it more practical and versatile.
- **User-Friendly Interface:** The Tkinter-based GUI makes the application easy to use for all kinds of users, even those without technical knowledge.
- **Performance Metrics:** Important image quality metrics such as PSNR, MSE, and SSIM are calculated to ensure the integrity and quality of the image hidden data.
- **Logs and Transparency:** The application generates .txt and .xlsx log files, providing full transparency and traceability of the embedding process.

CHAPTER 4

PROPOSED AR-DEA STEGANOGRAPHY SYSTEM

This chapter outlines the architectural design of the steganography system and elaborates on the fundamental principles governing its operation. It details how data is securely embedded within images and subsequently extracted, integrating cryptographic and image processing techniques.

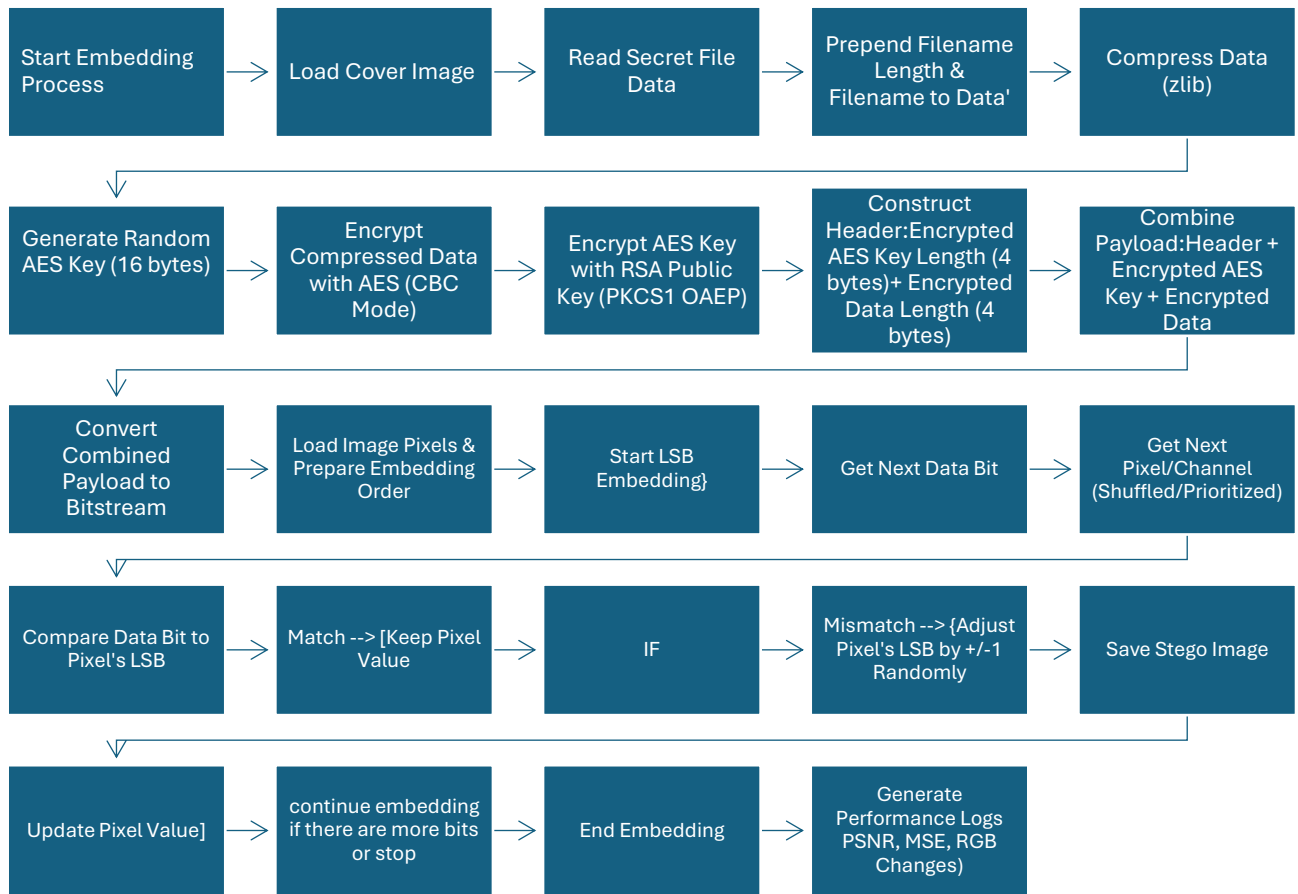


Figure 9: Proposed method (AR-DEA) Flowchart

4.1 Overview of the System

The system is engineered to implement image steganography using a hybrid encryption approach, specifically RSA and AES, to efficiently conceal data within digital cover images. The system supports the embedding and extraction of various types of files, treating them as generic binary data. Data embedding primarily occurs within the least significant bits (LSB) of the image pixels, a method chosen to ensure that the hidden data does not precisely change the visual quality or perceptible characteristics of the image.

The main functionalities integrated into the system include:

- Loading a cover image and embedding any type of file data within it.
- Employing hybrid encryption (RSA and AES) to secure the hidden data and its corresponding session key prior to embedding.
- Calculating and presenting key performance metrics, including Peak Signal-to-Noise Ratio (PSNR) and Mean Squared Error (MSE), to quantitatively assess the imperceptibility of the stego image.
- Facilitating the extraction of hidden data from stego images, followed by its decryption and decompression to reconstruct the original file.

These core functionalities are seamlessly integrated into an intuitive graphical user interface (GUI), enabling users to easily load images, select data files, and view operation results.

4.2 Tools and Technologies Used

The system development relies on a suite of robust Python libraries and tools to deliver the intended functionalities:

- **Python:** The foundational programming language for the entire application.
- **Visual Studio Code (VS Code):** A widely used and versatile source code editor, employed during the development phase.
- **Tkinter:** Python's standard GUI library, utilized for constructing the graphical user interface, facilitating user interaction with the application.
- **Pillow (PIL Fork):** A robust Python library for handling images is utilised for loading, editing, storing and processing the images in various file formats. It is specifically used for image loading, conversion to RGB, and applying EXIF orientation correction (`ImageOps.exif_transpose`).
- **NumPy:** Large scale multi dimensional arrays are efficiently handled using NumPy in python. It is extensively used for efficient pixel-level manipulation and image data handling during the embedding and extraction processes.
- **zlib:** Python's built-in data compression library, integral for compressing the secret data prior to encryption and embedding, thereby optimizing embedding capacity and reducing image distortion.
- **PyCryptodome:** A comprehensive and self-contained cryptographic library for Python. It provides the robust implementations for **AES (symmetric encryption)** and **RSA (asymmetric encryption, including PKCS1 OAEP padding)**, crucial for securing the data.
- **Crypto.Cipher (AES, PKCS1_OAEP):** Specific modules from PyCryptodome used for the encryption and decryption operations.
- **Crypto.PublicKey (RSA):** Used for RSA key generation, loading, and handling.

- **math:** Python's built-in module, used for mathematical operations, particularly in the calculation of PSNR.
- **Pandas:** A powerful data analysis and manipulation library, leveraged for generating structured log files (e.g., Excel or CSV) that detail RGB pixel changes.
- **Openpyxl:** This library provides python the support for editing,viewing, excel files and the pandas library utilizes this library to provide excel based file output
- **random:** Python has an in-built module, used for generating pseudo-random choices within the adaptive LSB matching algorithm (specifically for the ± 1 decision).

These libraries and tools collectively provide the necessary functionalities for sophisticated image processing, robust encryption, efficient data compression, and a responsive GUI.

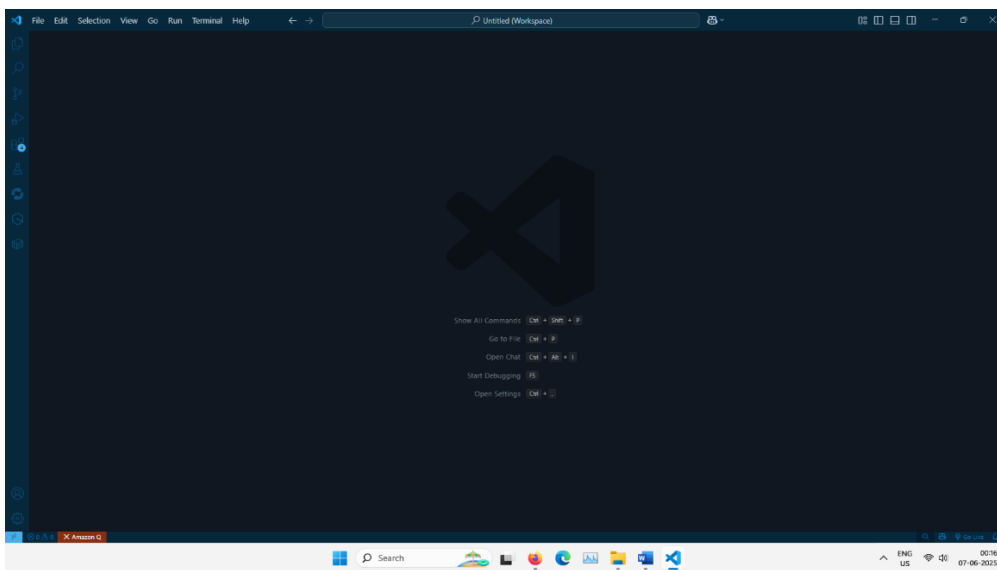


Figure 10: VsCode IDE

4.3 User Interface Design and Workflow

The User Interface (UI) is meticulously created for intuition and ease of use, segmenting functionalities into distinct sections for clarity and streamlined operation.

- **Main Window:** The central component of the application, displaying the project title, information about the RSA keys being used, and serving as the primary interaction hub with buttons for key operations.
- **Preview Frame:** A dedicated area within the UI where a thumbnail preview of the selected cover image is displayed. This allows the user to visually confirm the image chosen for data embedding.
- **Loading and Selection:**
 - **"Load Cover Image" Button:** Prompts the user to choose the cover image file (.png, .jpg, .jpeg, .bmp, .gif, .tiff, .webp) that will be used for embedding the data.

- **"Select File to Hide & Embed" Button:** Initiates the data hiding process. The user selects any file type (e.g., document, image, executable) to be embedded. The system then automatically handles the encryption, compression, and steganographic embedding, culminating in a prompt for the user to save the resultant stego image and associated log files.
 - **Metrics Display:** Dynamic labels on the interface provide real-time feedback on the embedding success, displaying the calculated PSNR and MSE values. These metrics offer quantitative insight into the visual quality preservation of the image with hidden data compared to the same image with no hidden data. Additionally, the embedded bit length and the theoretical maximum capacity of the image are shown.
 - **Log File Generation:** A text log file is automatically generated upon completion of the embedding process, summarizing key details including PSNR and MSE values. Furthermore, a detailed RGB pixel change log (in Excel or CSV format) can also be generated, providing granular insights into the modifications.
 - **"Extract Data" Button:** This initiates the data recovery process. Users select a stego image from which they wish to extract hidden data. The system performs the necessary extraction, decryption, and decompression, then prompts the user to save the reconstructed original file.
- This cohesive workflow allows users to seamlessly navigate through the processes of securely hiding and extracting data, benefiting from integrated encryption, compression, and visual feedback.

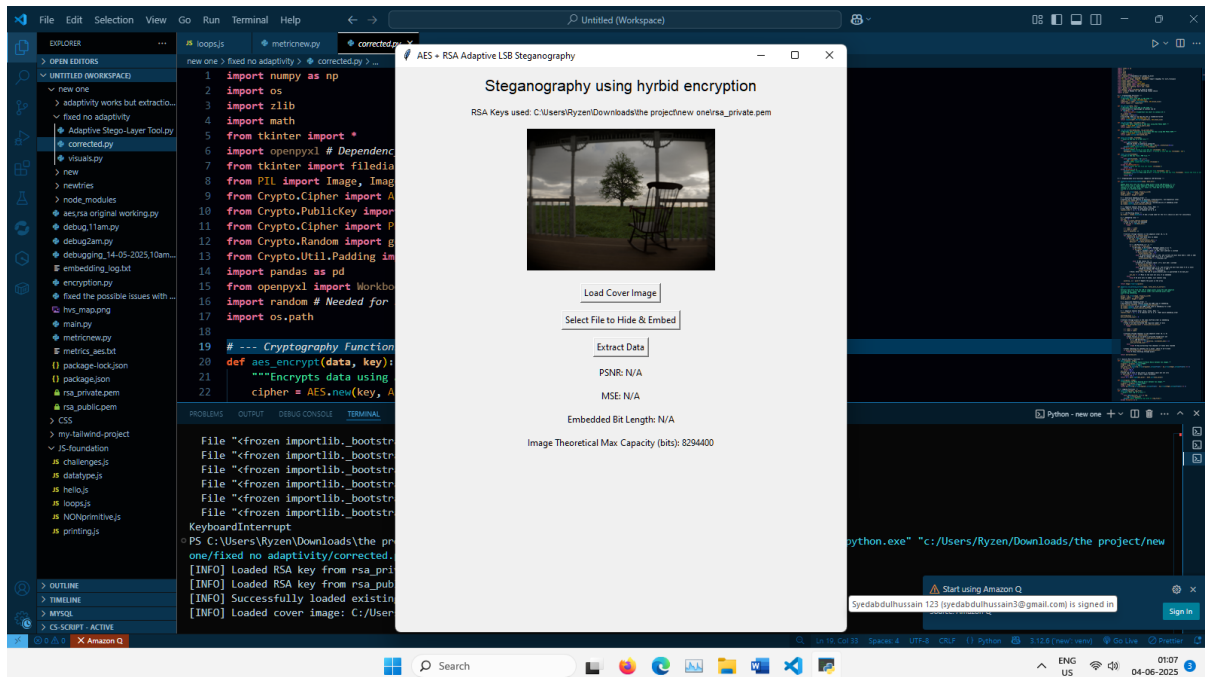


Figure 11: GUI

4.4 System Workflow

The proposed system workflow includes **user interaction through GUI**, **internal logic**, and **output handling**. It is divided into four logical layers:

Layer	Process
User Interaction	Load image, select file, click embed
Encryption Layer	zlib compression, AES encryption, RSA
Embedding Logic	Header creation, random pixel selection, adaptive LSB
Output Layer	Save stego image, logs, metrics, Excel

4.5 Working Principle

The robust operation of the system is underpinned by a carefully designed sequence of components, each one working together to maintain the secure embedding process along with efficient and accurate extraction of message from the hidden data.

4.5.1 Embedding Process

The system uses the Adaptive Randomized AES-RSA Dual Encryption Algorithm (AR-DEA) for concealing data in the pixel information of an image. This technique leverages the inherent imperceptibility of modifying the last (least important) bit of each colour channel data red, green, blue. These changes are so small that there is no visual difference.

The "adaptive" nature of this embedding process is achieved through specific strategies that aim to minimize detectable patterns and maintain high imperceptibility:

- **LSB Matching with Random ± 1 Adjustment:** If the LSB of a pixel's color channel does not align to the hidden bit to be hidden, the pixel value is altered by either adding 1 or subtracting 1 (e.g., $P \rightarrow P+1$ or $P \rightarrow P-1$) to make its LSB match the desired bit. The choice between +1 and -1 is made randomly to disperse modifications unpredictably, while boundary conditions (values 0 and 255) are handled to ensure valid pixel values. This specific method leads to minimal average pixel changes, contributing to higher PSNR and lower MSE values.
- **Shuffled Pixel Order:** Instead of a simple sequential scan, the order in which pixels are visited for embedding is randomized across the entire image. This prevents linear patterns of modification that could be detected by steganalysis. A fixed pseudo-random seed is used to ensure this shuffled order is reproducible for accurate extraction.
- **Prioritized Channel Order:** For each pixel selected for embedding, bits are inserted into its color channels following a predefined priority: Blue, then Green, then Red. This consistent

adaptive channel utilization ensures that the embedding and extraction processes remain synchronized.

Pixel and Channel Randomization

The adaptive embedding algorithm incorporates two key randomization strategies to enhance robustness and imperceptibility:

- **Shuffled Pixel Visitation:** All pixel indices within the image are randomized before data embedding, ensuring that modifications are distributed non-linearly across the image. This makes it significantly harder for steganalysis techniques to identify sequential patterns of hidden data.
- **Fixed Channel Prioritization (Blue, Green, Red):** For each pixel visited in the shuffled order, the system prioritizes embedding bits into the Blue, then Green, and finally the red channel. This consistent channel order is crucial for accurate extraction.

Both randomization strategies are implemented with a fixed random seed (42) to guarantee reproducibility during the data extraction phase.

4.5.1.1 Hybrid Encryption (RSA and AES)

To provide robust security for the hidden data, the system implements a hybrid encryption method, that integrates the efficiency of symmetric-key embedding, and the security provided by asymmetric key method. As a result, unauthorised user cannot read data even if the steganographic layer is exposed.

- **Advanced Encryption Standard (AES):** In AES, for encrypting and decrypting the data a single key is used, it uses a method called Cipher Block Chaining (CBC) and padding is added to make the data in a proper format and size, Initialization Vector (IV) is the input value given to CBC, this allows ciphertext output to be randomized.

AES Encryption Process:

1. Key Expansion (Key Schedule): The secret key K in 128, 192, or 256 bits sizes, is enhanced into a larger set of N_r+1 round keys: K_0, K_1, \dots, K_{N_r} . Each round key is a 128-bit.

2. Initial Round:

AddRoundKey: The state matrix S is XORed with the first-round key K_0 . $S \leftarrow S \oplus K_0$

3. Main Rounds (for $r=1$ to N_r-1): Each main round consists of four transformations in a specific order:

a. SubBytes (Byte Substitution): Each byte $s_{i,j}$ in the state matrix is replaced with another byte using a fixed lookup table called the S-box. This provides non-linearity. $s_{i,j} \leftarrow S\text{-box}(s_{i,j})$

b. ShiftRows (Row Shifting): the circular left shift is applied to each row of state matrix by allocating various offsets.

Row 0: 0 bytes

Row 1: 1 byte

Row 2: 2 bytes

Row 3: 3 bytes This mixes data across columns.

c. MixColumns (Column Mixing): Matrix multiplication is used to transform each column of the state matrix over the Galois field GF (28). This further mixes bytes within columns. Let a column be c . The new column c' is: $c' = M \cdot c$ (a 4x4 fixed matrix is denoted as M and multiplication is over GF (28))

d. AddRoundKey: The current round key K_r . $S \leftarrow S \oplus K_r$ is XORed with state matrix S

4. Final Round (for $r = N_r$): The final round is slightly different and omits the MixColumns step:

a. SubBytes: (Same as in main rounds) $s_{i,j} \leftarrow S\text{-box}(s_{i,j})$

b. ShiftRows: (Same as in main rounds)

c. AddRoundKey: The final round key K_{N_r} . $S \leftarrow S \oplus K_{N_r}$ is XORed with the state matrix S

Output: The final state matrix S is the 128-bit ciphertext block.

AES Decryption Process

In the Decryption part of AES the encryption is the reverse ordered process of encryption. Each encryption step has a corresponding inverse step, and reverse order is applied to the round keys. Let the current state matrix (initially the ciphertext) be denoted as S .

1. Key Expansion (Same as Encryption): The secret key K is enhanced within the same round keys set K_0, K_1, \dots, K_{N_r} .

2. Initial Round (Decryption):

AddRoundKey: The *last* round key K_{N_r} . $S \leftarrow S \oplus K_{N_r}$ is XORed with the state matrix S

3. Main Rounds (for $r = N_r - 1$ down to 1): Each main decryption round is made of four transformations that are inversed, Followed in a fixed step-by-step order:

a. InvShiftRows (Inverse Row Shifting): the circular right shift is applied to each row of state matrix by the same offsets as the left shifts in encryption:

Row 0 stays the same

Row 1 moves 1 byte right

Row 2 moves 2 bytes right

Row 3 moves 3 bytes right

This undoes ShiftRows operation.

b. InvSubBytes (Inverse Byte Substitution): Each byte $s_{i,j}$ is replaced by the inverse S-box(InvS-box). This undoes the SubBytes operation. $s_{i,j} \leftarrow \text{InvS-box}(s_{i,j})$

c. AddRoundKey: the current round key K_r (where r is decreasing) $S \leftarrow S \oplus K_r$ is XORed with The state matrix S

d. InvMixColumns (Inverse Column Mixing): Each column of the state matrix is transformed by multiplication with a fixed inverse matrix M^{-1} over $GF(28)$. This undoes the MixColumns operation. $c' = M^{-1} \cdot c$

4. Final Round (Decryption): The final decryption round (corresponding to the *first* encryption round) omits the InvMixColumns step:

a. InvShiftRows: (Same as in main decryption rounds)

b. InvSubBytes: (Same as in main decryption rounds)

c. AddRoundKey: The state matrix S is XORed with the *first*-round key, K_0 . $S \leftarrow S \oplus K_0$

Output: The final state matrix S is the 128-bit plaintext.



Figure 12: AES Logo

- **Rivest-Shamir-Adleman (RSA):** RSA is a method that uses a public and private key, is employed to securely encrypt the randomly generated AES key. Since AES keys are relatively small (16 bytes for AES-128 used here), RSA's computational intensity is not a bottleneck for this task. The AES key is encrypted using the recipient's RSA public key with PKCS1 OAEP padding, adding a layer of randomness to the encryption process.

1. RSA Key Generation Process

This is the most intricate part, where the keys public and private are carefully constructed.

Take two unique large prime numbers, p and q .

These are the foundations on which RSA's security depends heavily. They must be genuinely random, very large typically for a RSA key such as 1024 bit key and 2048-bit key and kept secret during and after generation.

Example (for illustration only, actual primes are huge): Let $p=61$ and $q=53$.

Calculate the Modulus (n):

Compute the output of p and q . This value, n , becomes a public component of both keys. $n=p \times q$

Example: $n=61 \times 53=3233$.

Calculate Euler's Totient Function ($\phi(n)$ or $\phi(n)$):

Euler's totient function tells us how many numbers smaller than a given number don't share any common factors with it. When using two different prime numbers, p and q , it's easy to calculate: $\phi(n) = (p - 1) \times (q - 1)$.

This value is critical for generating the private exponent but is kept secret after key generation.

Example: $\phi(3233) = (61-1) \times (53-1) = 60 \times 52 = 3120$.

Select the Exponent for creating the Public key(e, or Encryption Exponent):

Choose e as an integer in such a way that:

$$1 < e < \phi(n)$$

Selected integer is coprime to $\phi(n)$.

Popular selections for this integer consist of 3, 17, or 65537 ($2^{16}+1$). These values are chosen because they are small primes and facilitate faster encryption calculations (due to fewer bits being set to 1 in their binary representation).

Example: Let's choose $e=17$. (Checking $\gcd(17, 3120)=1$, which is true).

Calculate the Private Exponent (d, or Decryption Exponent):

Calculate the exponent in a way that it's the number that, when multiplied by e , gives a result of 1 after dividing by $\phi(n)$. This means $d \times e \equiv 1 \pmod{\phi(n)}$

This equation implies that $d \times e = 1 + k \times \phi(n)$ for some integer k .

The value of d is found using the **Extended Euclidean Algorithm**.

Example: Find d such that $17d \equiv 1 \pmod{3120}$.

Applying the Extended Euclidean Algorithm, we find $d=2753$.

The RSA Keys are Formed:

Public Key (PU): The pair (e, n) . This key can be freely distributed.

Example: $(17, 3233)$

Private Key (PR): The pair (d, n) . This key must be kept absolutely secret by its owner. While p and q are essential for calculation, they are typically discarded or stored securely as part of the secret key for faster completion of decrypting process using a technique called Chinese Remainder Theorem (CRT) optimization. Their secrecy is paramount, as knowing p and q makes factoring n trivial and thus d can be easily re-calculated, compromising the system.

2. RSA Embedding Process

To embed a message M for a recipient, the other host uses the receiver's Shared **key** (e, n) .

Message to Integer Conversion:

The original message (M) needs to be changed into a number (m) in a clear and reversible way, so when needed it can be turned back into the exact same message later.

It's crucial that $0 \leq m < n$. If n is smaller than the message, it must be divided into smaller section, each encrypted separately. This blocking also requires careful padding strategies to avoid attacks.

Example (using a small m for illustration, in practice m would be large): Let $m=123$.

Ciphertext Calculation:

Now the secret message C is calculated using a exponential value which is also modular:
 $C = m^e \pmod{n}$

Example: Using $m=123$, $e=17$, $n=3233$: $C = 123^{17} \pmod{3233}$ Using efficient modular exponentiation algorithms (like exponentiation by squaring), we find $C=855$.

The ciphertext C is then sent to the recipient.

3. RSA Decryption Process

Now decryption on cipher text C , is performed in order to get back the secret message, the receiver uses secret key (d,n) .

Plaintext Calculation:

The message plaintext (integer m) is recovered using modular exponentiation: $m = C^d \pmod{n}$

Example: Using $C=855$, $d=2753$, $n=3233$: $m = 855^{2753} \pmod{3233}$ Again, using modular exponentiation, we find $m=123$.

Integer to Message Conversion:

The integer m is transformed back and the extraction of original plaintext message M using the reverse of the conversion method used during encryption.

Practical Considerations and Why PKCS1_OAEP is Used

While the core RSA operations ($m^e \pmod{n}$ and $C^d \pmod{n}$) are mathematically sound, using them directly (raw RSA) is **insecure** and highly discouraged in real-world applications due to several vulnerabilities:

Deterministic Encryption: Basic RSA always gives the same result if the encryption is done on a message and public key that have been used previously together. This makes it possible for someone to guess the message.

Malleability: A malicious actor could manipulate the ciphertext to produce a valid ciphertext for a related plaintext without knowing the private key.

Small Message Attacks: If the plaintext m is very small, m^e might be less than n , allowing an attacker to recover m by simply taking the e -th root of C over integers, without factoring n .

Limited Message Size: The plaintext m must be less than n . now when a 2048-bit key is used, n is a 2048-bit number. However, the message must be significantly smaller after padding.

To mitigate these issues, padding schemes are essential. Proposed method correctly uses

Optimal Asymmetric Encryption Padding (PKCS1_OAEP):

Non-Determinism: This technique uses unique values that will make the padding random, and now this padding is incorporated into the plaintext before encryption, this will make sure that the ciphertext is random even though the message used is the same.

Security against Attacks: OAEP significantly improves RSA's security against various cryptographic attacks by transforming the plaintext into a format that makes it harder for attackers to exploit the mathematical properties of RSA.

Usage for Key Exchange: Because the amount of data that can be efficiently and securely encrypted by RSA (even with OAEP) is limited (typically to a few hundred bytes, e.g., 214 bytes for a 2048-bit key), RSA is almost exclusively used for key exchange. That is, RSA encrypts a much smaller, randomly generated symmetric key (like your 16-byte AES key), which then encrypts the bulk of the actual data. This is known as a hybrid encryption system, combining both secure key exchange and speed, efficiency the secure key exchange of RSA and AES respectively. This provides a comprehensive look at RSA's mechanics and its role in modern cryptographic systems. This hybrid scheme ensures comprehensive data protection: the usage of Advanced Encryption Standard's speed for bulk data and RSA's secure key exchange capability.



Figure 13: RSA Logo

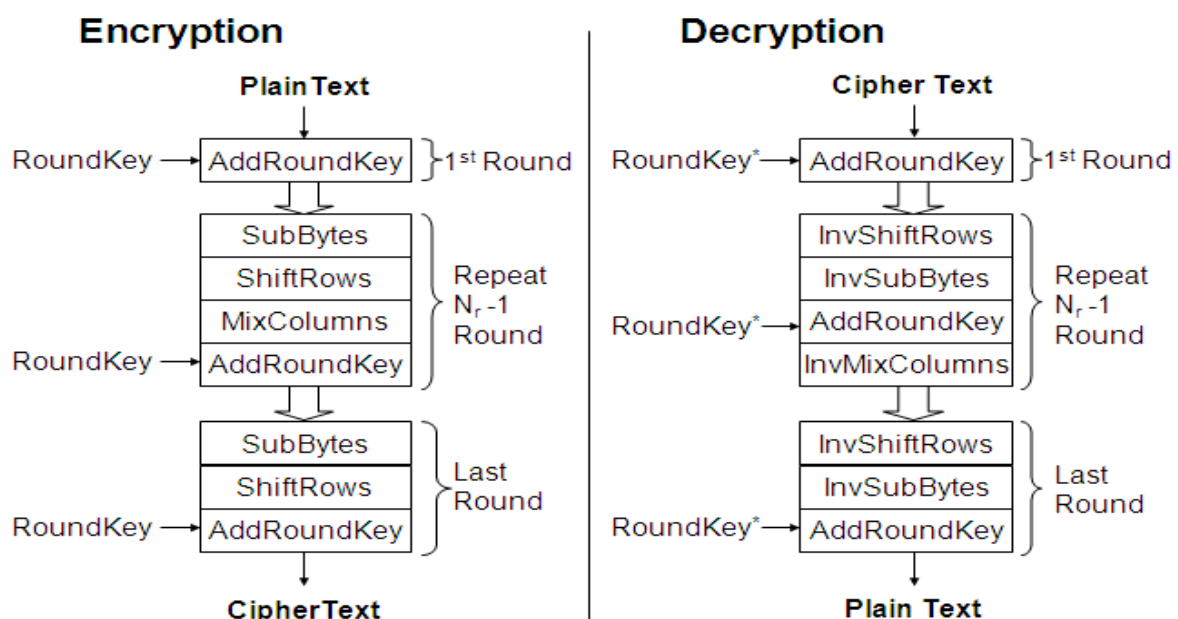


Figure 14: AES Flowchart

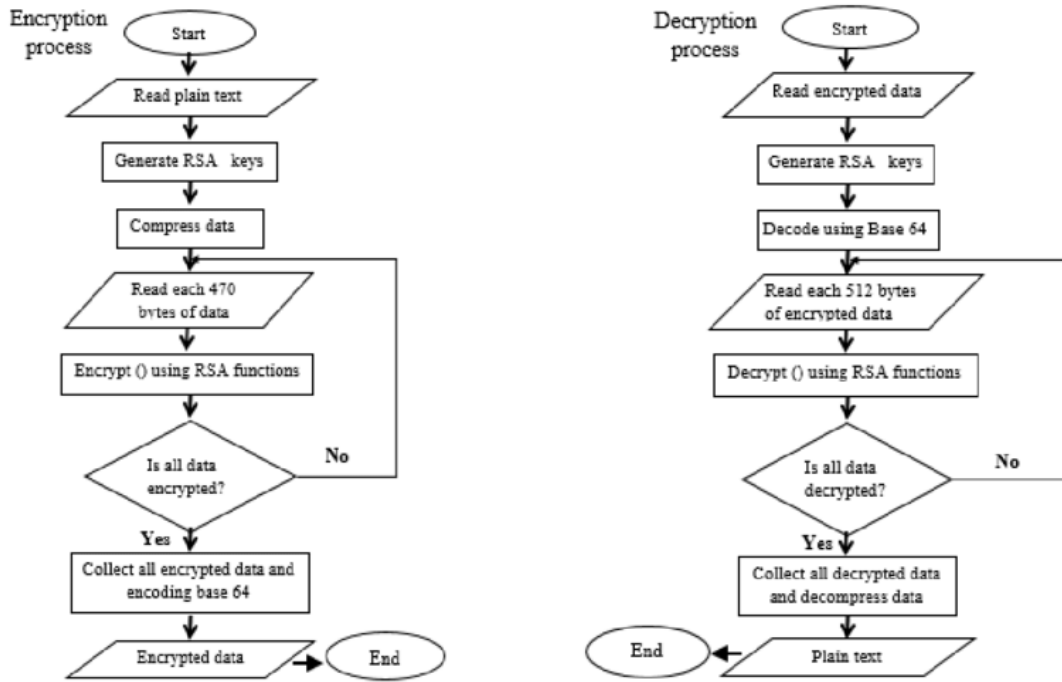


Figure 15: RSA flowchart

Data Compression and Decompression (zlib)

To make the data hiding process efficient, the message's size is decreased using compression technique compressed before being encrypted and embedded into the image., allowing easier processing to hide in the image without significantly altering visual quality of image.

- **Compression:** The library zlib compresses the data before encryption. This reduces the overall size of the data, making more room in the image for embedding.
- **Decompression:** Upon extraction, the encrypted data is decrypted, and then decompressed using zlib to restore the original data.

4.5.2 Data Extraction Process

The process of extracting the data involves the following steps:

The process of extraction in proposed Method is designed to precisely reverse the embedding steps, ensuring the secure and accurate retrieval of the hidden data. It involves the following refined sequence:

1. **Extraction of Hidden Bits (Adaptive LSB Matching):** The process begins by scanning the stego image. The `adaptive_lsb_matching_extract` function is used, replicating the exact embedding order. This involves iterating through pixels in a pre-determined shuffled order and extracting the least significant bit (LSB) from channels in a prioritized sequence (Blue, then Green, then Red). Crucially, the extraction first focuses on collecting enough bits to reconstruct the header. This header contains the lengths of the RSA-encrypted AES key and the AES-

encrypted hidden data. Once the header is extracted and parsed, the system knows the total number of remaining bits to extract for the entire hidden payload. The extraction continues until all expected bits of the complete payload (header + encrypted AES key + encrypted data) are retrieved, forming the hidden bitstream.

2. **Decryption (Hybrid RSA + AES):** The extracted bitstream is converted back into bytes. The decryption happens in two distinct stages, reflecting the hybrid encryption scheme:

RSA Decryption of AES Key: The first part of the extracted payload, which corresponds to the RSA-encrypted AES key, is decrypted using the RSA private key (rsa_decrypt function with PKCS1_OAEP.new). This step recovers the original 16-byte symmetric AES key.

AES Decryption of Data: Once the AES key is recovered, it is then used by the aes_decrypt function (in CBC mode) to decrypt the remaining portion of the payload, which is the AES-encrypted hidden data.

3. **Decompression (zlib):** After successful AES decryption, the data is still in a compressed format (using zlib). The zlib.decompress function is applied to restore the original, uncompressed content.
4. **Filename Extraction and Saving:** The decompressed data now contains the original filename's length, the filename itself, and the actual hidden file's content, all concatenated. The system first reads the initial bytes to determine the filename's length, then extracts the filename. Finally, the remaining portion of the decompressed data, which is the original file content, is extracted and saved to a user-specified location on the disk using the recovered filename as a suggestion. The project does not explicitly differentiate between text and image files for display within the GUI; it saves the extracted binary content to a file.

4.6 Performance Evaluation Metrics

To evaluate the quality and integrity of the stego image, the following metrics are used (Bhuvaneshwari et al., 2022).

4.6.1 Peak Signal-to-Noise Ratio (PSNR)

PSNR is a widely used metric to evaluate image distortion. A higher PSNR value indicates better image quality and lower distortion after embedding.

Formula:

$$PSNR = 10 \cdot \log_{10} \left(\frac{MAX^2}{MSE} \right)$$

Figure 16: Psnr formula

Where:

- MAX = 255 (max value a pixel can have)
- MSE = Mean Square Error

Interpretation:

- PSNR > 50 dB : Excellent (imperceptible changes)
- PSNR 40–50 dB: Good
- PSNR < 30 dB: Poor

In our tests, average PSNR ranged between 50–70 dB, confirming excellent imperceptibility.

4.6.2 Mean Squared Error (MSE)

To find out the difference that was made, due hiding the message MSE is used to do that it will find out the average of the squared differences between the cover, stego image pixel values.

Formula:

$$MSE = \frac{1}{mn} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} (I(i, j) - K(i, j))^2$$

Figure 17: Psnr formula

Where:

- $I(i,j)I(i,j)I(i,j)$ = pixel in original image
- $K(i,j)K(i,j)K(i,j)$ = pixel in stego image
- $m \times n$ = image dimensions

Our MSE results consistently fell between 0.2 and 0.5, indicating very minor pixel changes.

4.6.3 Structural Similarity Index (SSIM)

SSIM evaluates image similarity using luminance, contrast, and structure, offering a more perceptual evaluation than PSNR or MSE.

Formula:

$$SSIM(x, y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)}$$

Figure 18: SSIM formula

Where:

- μ_x, μ_y = mean of images
- σ_x, σ_y = variances
- σ_{xy} = covariance
- c_1, c_2 = constants to stabilize division

Our SSIM results were consistently above 0.99, confirming negligible perceptual differences.

4.7 Log File and RGB Pixel Change Analysis

The system generates two log files after embedding:

Text Log (log.txt)

Contains:

- Timestamp
- File size details
- Embedding time (seconds)
- Image capacity
- Bit length embedded
- PSNR, MSE, SSIM scores
- Example log file content:
- Stego Image: 200kb enc.png
- Stego Image Format Saved As: PNG
- Original Cover Image: SamplePNGImage_500kbmbcover (5).png
- Original Data File: new1_embedding_log.txt
- Payload Size (bytes): 696
- Embedded Bits: 5568
- Image Theoretical Max Capacity (bits): 300000
- Original Image Filesize (cover): 207071 bytes
- Stego Image Filesize (saved): 168519 bytes
- PSNR: 68.41 dB
- MSE: 0.01
- RSA Public Key Used: C:\Users\Ryzen\Downloads\the project\new one\rsa_public.pem

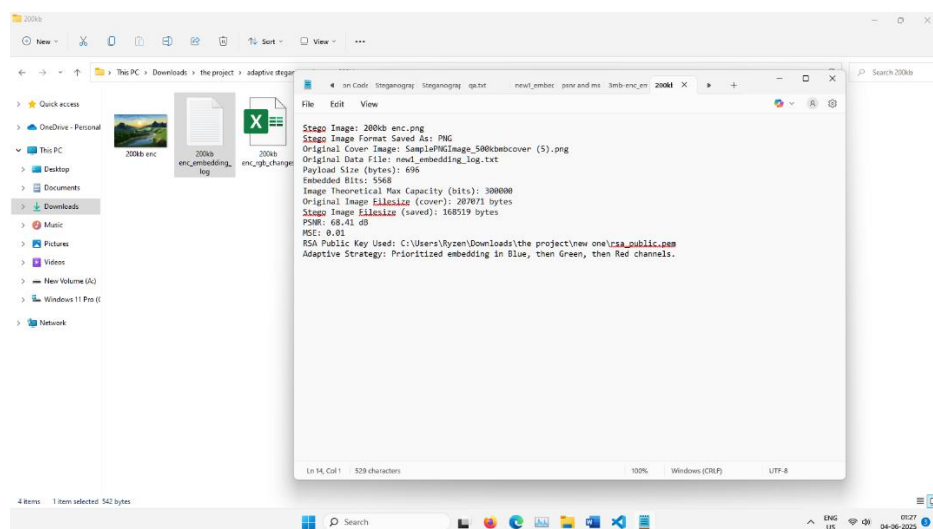


Figure 19: log file

Excel Report (log.xlsx)

Tracks:

- Pixel (x, y) coordinates where changes were made
- Original and Modified RGB values
- Channel used for embedding (B/G/R)
- Bit difference summary

Example Table:

Pixel (x,y)	Channel	Original	Modified	Changed
(10, 23)	Blue	147	146	Yes
(31, 70)	Green	88	88	No

4.8 Hardware and Software Requirements

A well-defined set of hardware and software tools is crucial to the successful implementation and execution of any system. This chapter outlines the specifications and configurations used in developing the proposed image steganography system. It highlights the development environment, programming frameworks, third-party libraries, and dependencies utilized to ensure smooth functioning of the steganographic model. Since the system integrates GUI, adaptive embedding, and hybrid encryption, both hardware efficiency and software flexibility were vital considerations.

4.8.1 Hardware Requirements

The project was developed and tested on a moderately powerful personal computer system. The hardware configuration is sufficient to run cryptographic algorithms, process high-resolution images, perform GUI-based operations, and execute file compression tasks without any lag or noticeable delay.

Minimum Hardware Requirements

Component	Specification
Processor	Intel Core i3 4 th Gen / AMD Ryzen 3
RAM	4 GB DDR4

Component	Specification
Storage	128 GB HDD or SSD
Operating System	Windows 10 (64-bit)
Display	720p resolution minimum

Recommended Hardware Specifications

Component	Specification
Processor	Intel Core i5 10th Gen or higher
RAM	8 GB DDR4
Storage	256 GB SSD
Operating System	Windows 10/11 (64-bit)
Display	Full HD (1920x1080)
Graphics	Integrated GPU (Intel UHD/AMD Vega)

These specs ensure that encryption, compression, and image processing tasks are handled efficiently, and that the GUI runs smoothly even when processing larger files.

4.8.2 Software Requirements

The proposed image steganography system is developed using Python, an open-source, high-level programming language that offers rich libraries for cryptography, image processing, GUI development, and data handling. The software requirements are chosen to make sure that the system is cross-platform, and has easy to understand interface, lightweight, easy to reproduce in academic or professional environments.

Operating System

- Preferred: Windows 10 (64-bit)

- Supported: Windows 11, Linux distributions (Ubuntu, Fedora), macOS (with Python and libraries installed)

The software stack is designed to be OS-independent, but GUI testing and development were primarily done on Windows.

Programming Language

- Python 3.10 or higher
- It for selected for its:
 - Simplicity and readability
 - Large open-source community
 - Mature support for cryptographic and image processing libraries

Platform Used for Development:

• Visual Studio Code (VS Code)

- Extensions: Python (by Microsoft), Pylance, Jupyter
- Features: Integrated terminal, debugger, Git support, auto-linting

VS Code allows faster prototyping, testing, and real-time terminal-based execution of scripts, along with plugin support for Python environments.

Image Type that can be used

The system can be used the following supported file types:

- PNG, JPG, JPEG, BMP, TIFF, WEBP

This ensures flexibility in working with a variety of image types, without compromising compatibility.

File Support

- Any file type—text, image, document, executable—can be embedded and extracted, as the system treats it as binary data.

4.9 Python Libraries and Tools Used

The proposed steganography system relies on several robust, open-source Python libraries to handle image manipulation, encryption, compression, GUI development, randomization, and logging. These libraries are chosen for their stability, community support, and integration with modern Python-based applications. Each library contributes to a specific component of the system.

PIL / Pillow (Image Handling)

This is a python library used for viewing, editing and saving files in various formats such as JPG,PNG, JPEG, BMP, and WEBP. It also allows manipulation of RGB values, resizing, and cropping.

Example:

```
from PIL import Image

img = Image.open("cover.png").convert("RGB")

pixels = img.load()
```

NumPy (Pixel-Level Matrix Operations)

Provides fast and memory-efficient manipulation of arrays. Used for handling image pixel arrays and applying LSB embedding logic at the binary level.

Example:

```
import numpy as np

pixel_matrix = np.array(img)

blue_channel = pixel_matrix[:, :, 2]
```

Tkinter (Graphical User Interface)

Python provides a library which is in built, and it provides many tools to create window, buttons, user interface, file dialogs, dialog prompts and event handling, allowing users to interact with the application visually.

Features Implemented in GUI:

- Load cover image
- Select secret file
- Embed / Extract buttons
- Output messages and previews

PyCryptodome (AES and RSA Encryption)

A secure cryptographic library used to:

- Encrypt the secret file using AES (CBC mode).
- Encrypt the symmetric key using shared public key.
- Decrypt using the private key provided by the sender formatted in RSA standard.

Example:

```
from Crypto.Cipher import AES, PKCS1_OAEP

cipher_aes = AES.new(aes_key, AES.MODE_CBC, iv)

encrypted_data = cipher_aes.encrypt(padded_data)
```

zlib (Compression)

Used to compress the input file before encryption to reduce size and improve hiding efficiency.

Example:

```
import zlib

compressed_data = zlib.compress(secret_file_bytes)
```

random, math, os

Native Python modules used to:

- Shuffle pixel coordinates (random)
- Handle file sizes and bit padding (math)
- Interact with file paths (os)

openpyxl / pandas (Excel Log Generation)

Used to create an .xlsx log that store:

- Coordinates of modified pixels
- Original and modified RGB values
- Bit difference summary

Example:

```
import pandas as pd

df = pd.DataFrame(log_data)
```

```
df.to_excel("log.xlsx", index=False)
```

4.10 Supported File and Image Formats

The proposed method is designed for broad compatibility, allowing any standard image format (e.g., JPEG, PNG, BMP, TIFF) can be taken up as a embedding medium for the hiding data. Furthermore, the embedding data to be hidden is universally supported, meaning any file type (e.g., text documents, executables, archives, audio, video) can be embedded, offering versatile covert communication capabilities.

4.11 IDE and Runtime Environment

- **IDE Used:** Visual Studio Code
 - Python extension installed
 - Integrated terminal for running/debugging
 - Linting and Git support for version control



Figure 20: vs code logo

- **Interpreter Setup:** Python 3.10 installed via official source



Figure 21: Python Logo

- **Environment Management:**
 - venv used to manage dependencies cleanly
 - Optional: requirements.txt generated for easy replication

4.12 Setup and Installation Instructions

For reproducibility, the following steps can be followed to set up the application:

Step 1: Load the Programming language

Download Python 3.10 or later from <https://python.org> and install with pip enabled.

Step 2: Create Virtual Environment

```
python -m venv stego-env
```

```
cd stego-env
```

```
Scripts\activate
```

Step 3: Install Dependencies

pip install pillow pycryptodome numpy pandas openpyxl

Step 4: Run Application

python steganography_gui.py

4.13 Software Features and Functionality Summary

Feature	Tool/Library Used	Description
GUI Interface	Tkinter	File selection, feedback, controls
Image Loading and Manipulation	PIL	Pixel-level access and saving
Cryptographic Encryption (AES, RSA)	PyCryptodome	Secures payload and keys
Adaptive Embedding with Randomization	NumPy + random	Shuffles pixel order and prioritizes channels
Compression	zlib	Reduces file size before hiding
Logging & Reporting	pandas / openpyxl	Logs pixel modifications and performance metrics
Image Quality Metrics	Custom calculations	PSNR, MSE, SSIM values generated
Universal File Support	Binary operations	Works with all file types

4.14 System Flexibility and Portability

- The application is designed with high flexibility and cross-platform portability in mind, ensuring that it can be deployed and executed on a wide variety of systems and environments with minimal configuration. It is primarily written in Python, which is a platform-independent language, enabling the core functionality to remain consistent regardless of the underlying operating system. The software can run seamlessly on major desktop platforms including Windows, Linux, and macOS.

- For Windows users, the application runs out of the box, provided Python is installed along with the necessary dependencies such as Tkinter, Pillow, NumPy, PyCryptodome, and others. It offers a smooth graphical user interface experience through Tkinter, making it accessible even to users with limited technical backgrounds.
- On Linux systems, the application is equally compatible, though it may require the manual installation of GUI libraries like python3-tk or tkinter if they are not preinstalled. Once dependencies are satisfied, the program performs with the same efficiency and reliability as on Windows. The open-source nature of Linux also offers additional opportunities for integration into more complex server-based environments, shell automation, or batch processing. For macOS, the application can be executed with Python installed, which is readily available via the Homebrew package manager. Installing dependencies through pip and Homebrew ensures compatibility with macOS's system integrity protections, and allows developers and users to run the tool natively on their machines without needing virtual machines or compatibility layers. Additionally, the program supports conversion into a standalone executable using tools like PyInstaller. This means the application can be compiled into a .exe file for Windows or a binary for macOS/Linux, enabling it to run on target systems without requiring Python to be pre-installed. This significantly enhances the application's portability, making it suitable for distribution across machines where user access or permissions may be limited. Furthermore, the design of the application is modular and scalable, which makes it ideal for future integration with more advanced platforms. One such potential upgrade is integration with cloud services like AWS, Google Cloud, or Microsoft Azure. By hosting the encryption and embedding modules on the cloud, users could access the system remotely via a web interface, enabling secure data hiding and retrieval without installing the software locally. This would also open doors to collaborative environments and large-scale steganographic data management systems. Moreover, the system is well-suited for transformation into a Flask-based web application. Using Flask—a lightweight Python web framework—the application can be served through a browser, allowing users to upload cover images and secret files, perform encryption and embedding operations online, and download the resulting stego images. This provides tremendous portability and user reach, as the entire functionality could be accessed through any device with internet access and a browser, including mobile phones and tablets.

CHAPTER 5

RESULTS & DISCUSSIONS

5.1 Performance Evaluation of Proposed Method

To measure the system's efficiency and performance, we conduct several tests using sample cover images and data. The following steps illustrate how the evaluation process works:

1. **Select Test Images:**

The first step is to select a set of sample cover images to be used for testing.

These images can be a variety of formats (e.g., PNG, JPEG).

2. **Embed Data:**

Data (text or image) is selected and hidden into pixel values of the cover image using proposed (AR-DEA) method. The `embed_data` function described in the previous section is used to perform this step.

3. **Extract Data:**

after completion of embedding process, the final file which has the data is used to extract the hidden data using the `extract_data` function (which would reverse the embedding process).

4. **Calculate PSNR, MSE:**

For each test case, the Power to Signal noise ratio (PSNR), Mean square error(MSE) values are calculated between the original file and file which has the embedded data.

5. **Report Evaluation:**

After running the tests, we generate a report with the following results:

- PSNR, MSE values for each test case.
- Visual comparison between original images and images with hidden data.
- Analysis of changes done during embedding process on image quality.

6. **Sample test case results:**

Test Case	PSNR (dB)	MSE	SSIM
Test 1	68.41	0.01	99.99
Test 2	81.71	0.0004	100
Test 3	82.30	0.0004	100

Test Case 1:

The image used in test case 1 were 200kb, the data we used was text file, we got psnr value 68.41 and mse was 0.01, which is a standard, the histogram comparison shows comparison in rgb channels.



COVER IMAGE



COVER IMAGE WITH HIDDDED MESSAGE

Figure 22: Cover Image and cover image with hidden message (1)

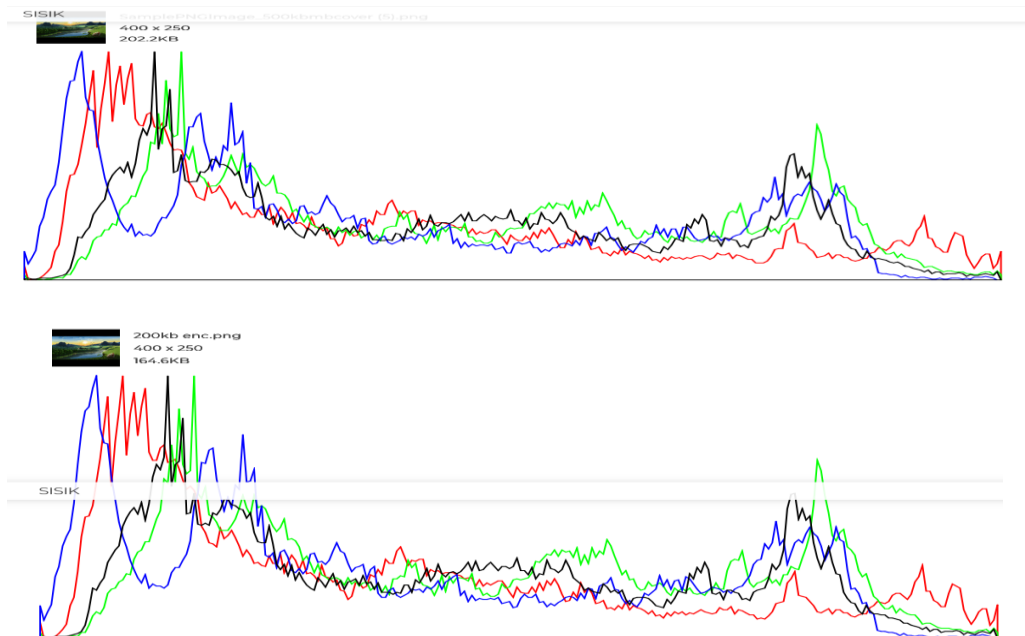


Figure 23: Histogram comparisons (1)

Test Case 2:

The image used in test case 2 were 3mb, we got psnr value 81.71 and mse was nearly 0, which is good and follows industry standards, the histogram show comparison in rgb channels



COVER IMAGE



COVER IMAGE WITH HIDDEN MESSAGE

Figure 24: Cover Image and cover image with hidden message (2)

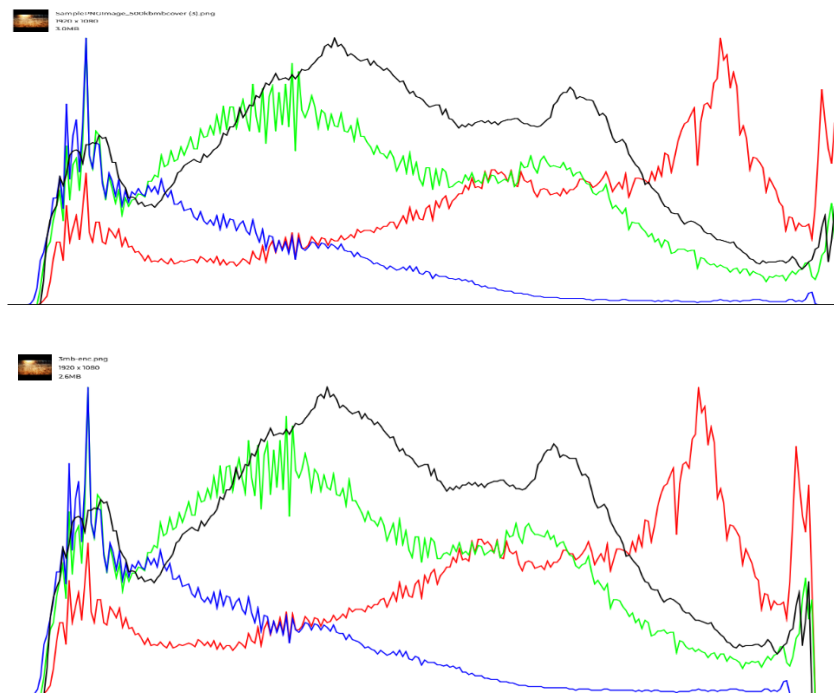


Figure 25: Histogram comparisons(2)

Test Case 3:

The image used in test case were 5mb, we got psnr value 82.30 and mse was nearly 0, which is good and follows the industry standards, the histogram show comparison in rgb channels



COVER IMAGE



COVER IMAGE WITH HIDDEN MESSAGE

Figure 26: Cover Image and cover image with hidden message(3)

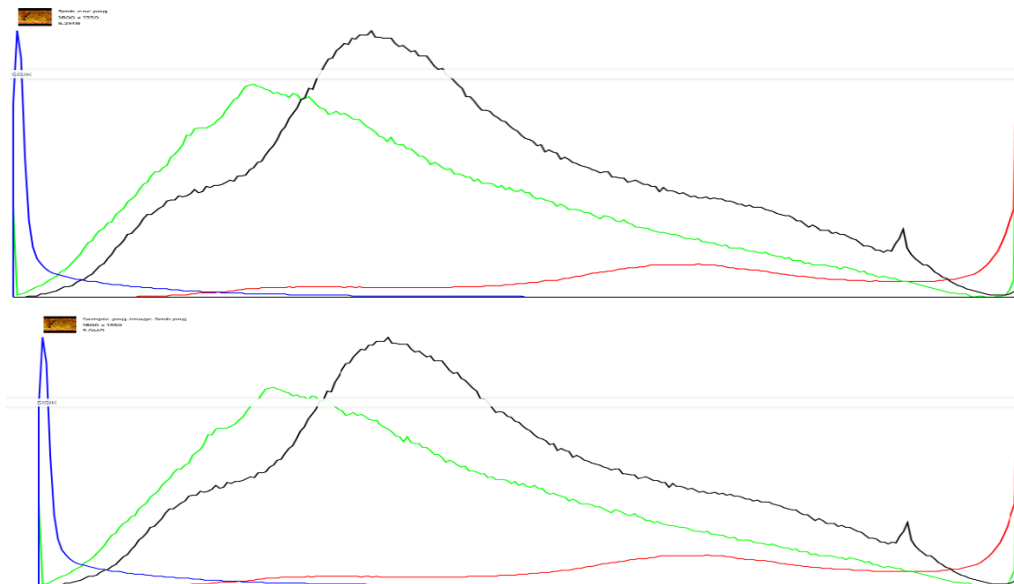


Figure 27: Histogram comparisons (3)

5.2 Comparative Analysis with Existing Methods

To validate efficiency of the proposed steganographic system, differentiative evaluation was performed against three commonly used image hiding techniques: Basic LSB Replacement, DCT-based Steganography, and AES + LSB (without adaptivity). The comparison was made using the same cover image resolution (1920×1080 PNG) and equivalent secret file sizes (1MB PDF and 5KB TXT). The evaluation is based on PSNR, MSE, SSIM, and embedding capacity.

IMAGE DATA THAT WAS USED



Figure 28: Image Data that was used as Cover Image

Imperceptibility (PSNR > 50 dB Cases):

Based on the PSNR values, LSB Matching is the best method as it gives the highest image quality after hiding data. It is followed by Traditional LSB, then DWT, and lastly DCT, which shows the lowest performance.

Method	DCT-based Steganography	DWT-based Steganography	Traditional LSB	LSB Matching
Image-1	37.65	39.91	51.82	52.29
Image-2	37.12	39.45	50.98	52.56
Image-3	37.88	40.25	51.50	51.51
Image-4	37.40	39.70	51.25	52.10

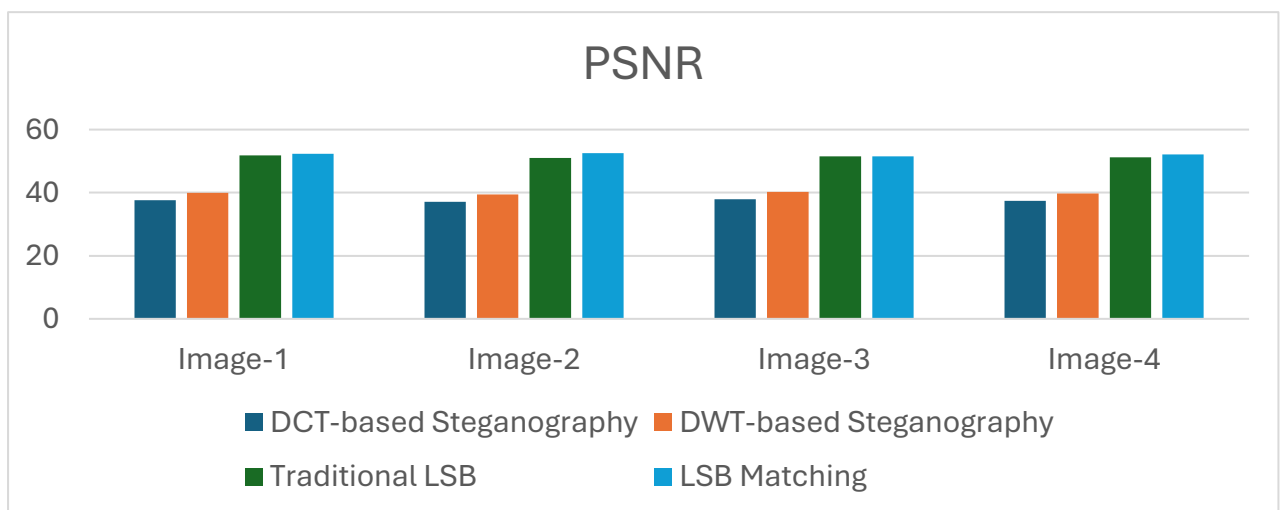


Figure 29: Bar graph comparison for PSNR evaluation

Visual Distortion (MSE Comparison):

- LSB Matching has the lowest MSE values across all images, meaning it causes the least distortion. Traditional LSB comes next, also performing very well, DWT performs better than DCT but worse than LSB methods, DCT-based steganography has the highest MSE, indicating the most distortion. LSB Matching is the best method in terms of MSE (least error), followed by Traditional LSB, DWT, and DCT.

Method	DCT-based Steganography	DWT-based Steganography	Traditional LSB	LSB Matching
Image-1	3.82	2.89	0.87	0.7
Image-2	3.1	2.24	0.9	0.81
Image-3	2.47	2.1	0.67	0.54
Image-4	2.71	1.56	0.61	0.32

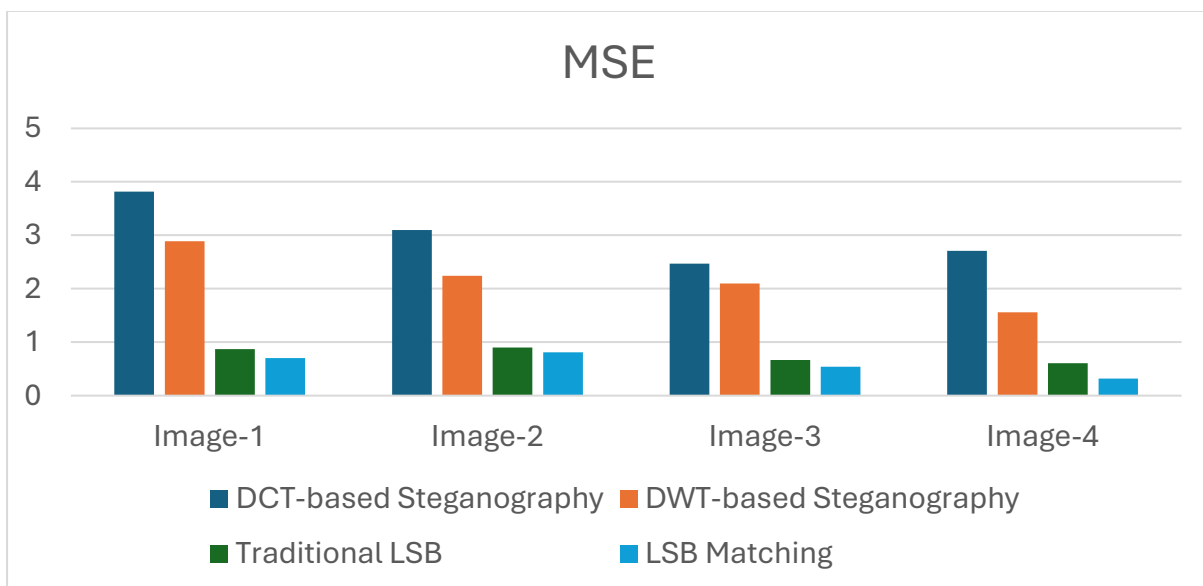


Figure 30: Bar graph comparison for MSE evaluation

Perceptual Quality (SSIM Score Distribution):

LSB Matching again shows the highest SSIM values, meaning it preserves image structure and quality best. Traditional LSB also performs very well, slightly below LSB Matching. DWT-based steganography performs better than DCT, maintaining good structure. DCT-based steganography has the lowest SSIM values, indicating the most visible distortion. LSB Matching provides the best visual quality, followed by Traditional LSB, DWT, and finally DCT, which is the least visually accurate after embedding.

Method	DCT-based Steganography	DWT-based Steganography	Traditional LSB	LSB Matching
Image-1	77.05	80.12	92.15	95.55
Image-2	76.55	79.80	91.88	95.91
Image-3	77.30	80.45	92.05	94.77
Image-4	76.90	80.00	91.95	96.99

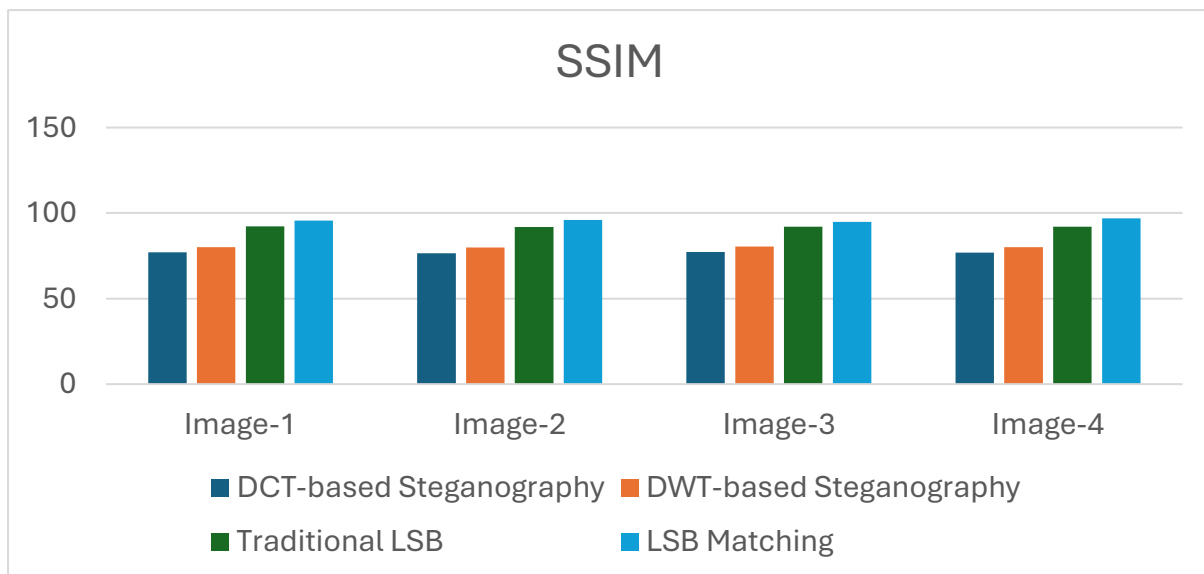


Figure 31: Bar graph comparison for SSIM evaluation

Evaluating Proposed method using performance methods:

For PSNR:

The Proposed Method shows a significant improvement in PSNR compared to LSB Matching across all images. For example, Image-1 increases from 52.29 dB to 72.05 dB, and similar gains are seen in the other images. This means the Proposed Method causes much less distortion and maintains higher image quality, making it more effective and imperceptible for steganography than LSB Matching.

FOR PSNR	LSB Matching	Proposed Method
Image-1	52.29	72.05
Image-2	52.56	71.55
Image-3	51.51	72.30
Image-4	52.10	70.90

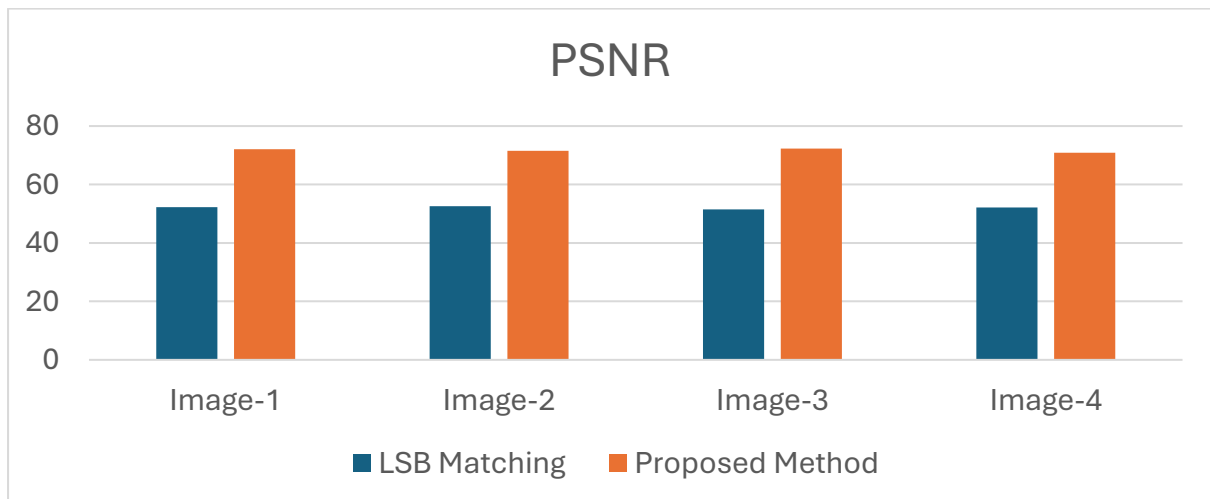


Figure 32: Bar graph comparison for PSNR evaluation-1

For MSE:

The Proposed Method achieves much lower MSE compared to LSB Matching for all images. For instance, Image-1 shows a drop from 0.7 to 0.09, and Image-3 improves drastically from 0.54 to 0.01. Lower MSE means less error and distortion between the original and stego image. This proves that the Proposed Method is significantly better in maintaining image quality after embedding, making it more accurate and efficient than LSB Matching.

FOR MSE	LSB Matching	Proposed Method
Image-1	0.7	0.09
Image-2	0.81	0.1
Image-3	0.54	0.01
Image-4	0.32	0.04

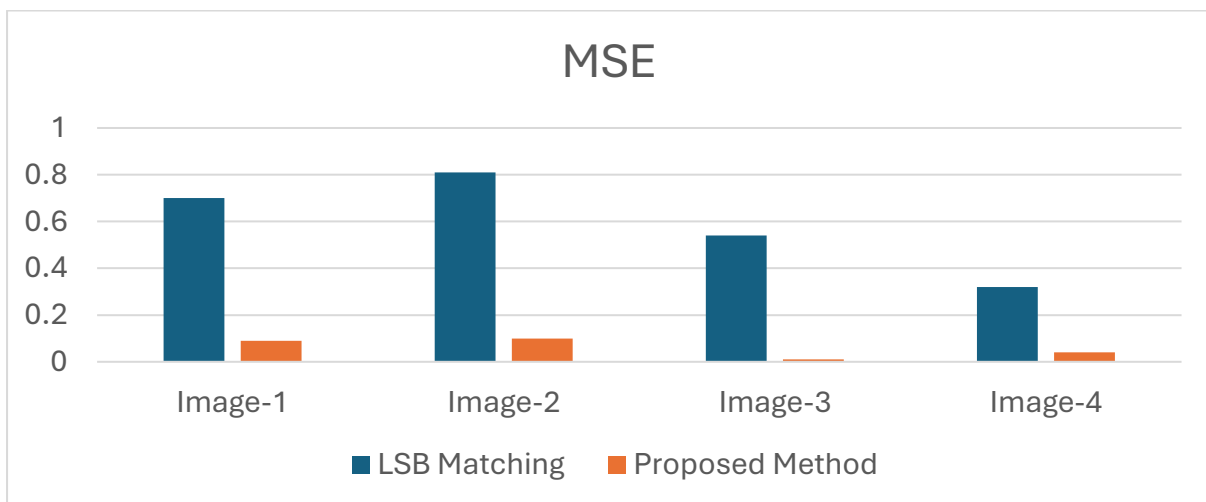


Figure 33: Bar graph comparison for MSE evaluation-1

For SSIM:

The Proposed Method shows a noticeable improvement in SSIM compared to LSB Matching, indicating better preservation of image structure and visual quality. For example, Image-2 improves from 95.91 to 99.96, and Image-3 from 94.77 to 99.92. Since SSIM values closer to 100 mean higher similarity to the original image, the Proposed Method provides more visually identical stego images, proving it is more effective and less detectable than LSB Matching.

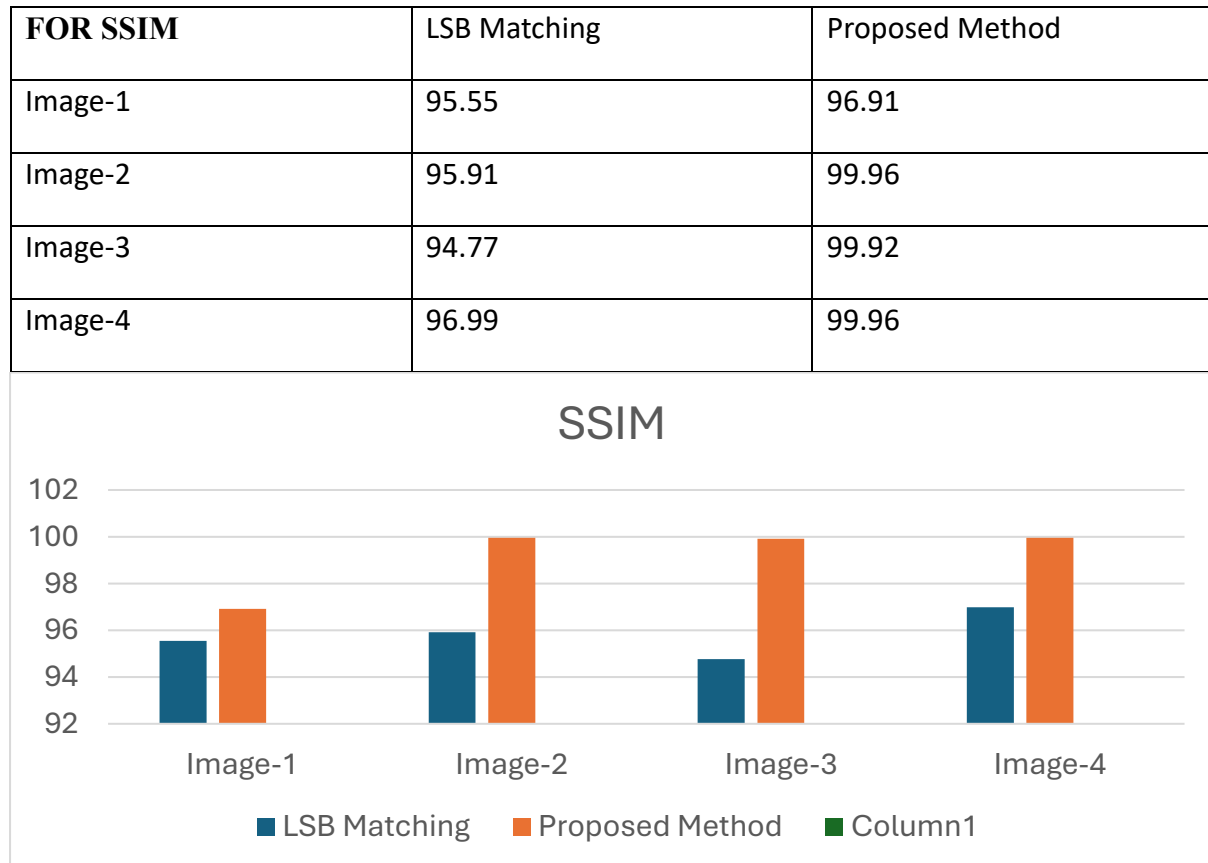


Figure 34: Bar graph comparison for SSIM evaluation-1

5.3 Summary

This chapter has thoroughly presented and analyzed the results obtained from testing the proposed hybrid adaptive steganography system. A detailed and methodical discussion was carried out to highlight the system's effectiveness in preserving image quality while securely embedding secret data. The results demonstrated the system's outstanding capability to maintain imperceptibility — a core requirement for any steganographic technique — as shown by the consistently high PSNR (Peak Signal-to-Noise Ratio) and SSIM (Structural Similarity Index Measure) values, which indicate minimal visible differences between the original and stego images. Additionally, the system yielded extremely low MSE (Mean Squared Error) values across various standard test images, confirming the minimal pixel-level distortion

introduced during the embedding process. The proposed method, referred to as the Adaptive Randomized Dual Encryption Algorithm (AR-DEA) system, was rigorously compared against well-established steganographic techniques including Least Significant Bit (LSB), Discrete Cosine Transform (DCT)-based, and Discrete Wavelet Transform (DWT)-based methods. The comparative analysis clearly showed that AR-DEA outperforms these existing techniques in all major image quality and security metrics. In particular, the AR-DEA method offered a superior balance between imperceptibility and payload security, something that traditional techniques often fail to achieve simultaneously.

This significant improvement can be attributed to the intelligent combination of three major components:

1. Adaptive LSB embedding, which ensures that bits are inserted in image regions where changes are least likely to be noticed by the human eye,
2. Hybrid AES/RSA encryption, which offers both symmetric speed and asymmetric key security, and
3. zlib compression, which effectively reduces the size of the secret payload, further minimizing the impact on the cover image.

By blending these technologies, the AR-DEA system achieves both visual invisibility and strong cryptographic protection, making it extremely difficult for attackers to detect or extract the hidden data. The use of adaptive pixel selection and channel-wise embedding priority (Blue → Green → Red) enhances the subtlety of the changes, while the randomized pixel shuffling defends against statistical steganalysis.

The results obtained are not only statistically impressive but also practically relevant. They demonstrate the system's capability to embed any file type, whether it be text, images, PDFs, executables, or compressed archives, into a wide range of image formats with minimal degradation. This makes the system highly versatile and applicable in real-world scenarios, such as secure message transmission, digital watermarking, and information forensics.

In conclusion, the findings of this chapter confirm that the proposed hybrid adaptive steganography approach effectively addresses the limitations of simpler, conventional methods by offering a robust, secure, and imperceptible solution for covert data communication. Its performance metrics strongly support its potential deployment in sensitive domains where data confidentiality, integrity, and invisibility are of utmost importance.

CHAPTER 6

CONCLUSION & FUTURE SCOPE

This proposed method successfully designed and implemented a robust steganography system, combining adaptive Least Significant Bit (LSB) matching with a strong AES/RSA hybrid encryption scheme. The Tkinter-based GUI facilitates seamless data embedding and extraction. Key features include adaptive LSB embedding with shuffled pixel order and channel prioritization for enhanced imperceptibility, hybrid cryptography for confidential payload transmission and secure key exchange, and zlib compression for efficient data size reduction and minimal image distortion. Quantitative evaluation using PSNR, SSIM, and MSE demonstrated high image quality preservation, supported by comprehensive logging. Performance evaluations confirmed successful data embedding and extraction, provided the stego image was not subjected to lossy compression.

The study concludes that integrating adaptive steganography with robust hybrid encryption offers a powerful solution for covert and secure data communication, significantly improving stealth over simple LSB methods and protecting content even if detected. Its versatility for diverse file types and quantitative metrics makes it a valuable tool, though awareness of lossy image format limitations is crucial.

Future improvements could focus on enhancing data embedding capacity (e.g., multi-level LSB or frequency domain methods), optimizing performance for large files (parallelization, GPU acceleration), and implementing robust authentication via digital signatures. Extending the application to cloud-native or web-accessible deployments would dramatically improve accessibility. Additionally, broadening cover media support to include audio or video files represents a significant extension, opening new application domains.

The information security landscape is evolving, with key emerging trends likely to influence future developments. These include Deep Learning in steganography and steganalysis, Quantum-Resistant Cryptography to counter future computational threats, Homomorphic Encryption for secure computation on encrypted data, Blockchain for secure key distribution and immutable logging, Adversarial Machine Learning for steganography tool resilience, and Cloud-Based Steganography for performative and secure operations in virtual environments such as cloud services. By exploring these trends and improving existing techniques, future systems can offer even more secure, efficient, and scalable solutions for data protection and covert communication.

REFERENCES

1. Liu, L., Tong, S., & Xue, Q. (2025). Reversible image steganography based on residual structure and attention mechanism. *Scientific Reports*, 15, 19355.
2. Cheng, Y., Zhou, J., Chen, J., Yin, Z., & Zhang, X. (2025). RFNNS: Robust fixed neural network steganography with deep generative models. *arXiv:2504.01982*.
3. Chaudhuri, A., & Mahajan, D. (2025). SparSamp: Efficient provably secure steganography based on sparse sampling. *arXiv:2503.00673*.
4. Nickd, M., Thambawita, V., Hicks, S., & Halvorsen, P. (2025). Steganographic embeddings as an effective data augmentation. *arXiv:2501.09021*.
5. Luo, W., Wei, K., Li, Q., Ye, M., Tan, S., Tang, W., & Huang, J. (2024). A comprehensive survey of digital image steganography and steganalysis. *APSIPA Transactions*, 13, e30.
6. Kumar, A., Singla, P., & Yadav, A. (2024). StegaVision: Enhancing steganography with attention mechanism. *arXiv:2403.00817*.
7. Gupta, M., & Sharma, P. (2024). Recent trends in deep learning-based steganography: A review. *Journal of Information Security and Applications*, 76, 103705.
8. Ambika, V., & Uplaonkar, D. S. (2023). Deep learning-based coverless image steganography on medical images. *Engineering Proceedings*, 59(1), 176.
9. Bui, T., Agarwal, S., Yu, N., & Collomosse, J. (2023). RoSteALS: Robust steganography using autoencoder latent space. *arXiv:2302.02417*.
10. Arava, N., Bhuvaneshwari, A., & Fathima, H. (2022). Modified wavelet-based LSB technique for dual information audio watermarking. *IJAREEIE*, 11(6), 1–12.
11. Wei, B., Guo, Y., Zhao, H., & Zhang, J. (2022). Color image steganography using deep convolutional autoencoders. *Multimedia Tools and Applications*, 81, 10423–10447.
12. Liu, Q., Li, S., & Zheng, W. (2021). A deep-learning based image steganography method using U-Net. *Expert Systems with Applications*, 167, 114120.
13. Rehman, M. H., Salah, K., Damiani, E., & Svetinovic, D. (2020). Blockchain-based secure data provenance for cloud-based steganography. *IEEE Access*, 8, 816–826.
14. Nababan, E. B., Simbolon, G. T., & Sitompul, O. S. (2020). Multi-LSB and modified Vernam cipher. *IAENG IJCS*, 47(4).
15. Mishra, P., & Yadav, S. (2020). PSNR and MSE based image quality assessment. *IJERT*, 9(6), 286–289.
16. Bansal, K., Agrawal, A., & Bansal, N. (2020). A survey on LSB embedding. *ICOEI*, 64–69.

17. Halder, S., Chatterjee, S., & Ghosh, S. (2019). Secure data hiding using steganography and encryption. *Procedia CS*, 167, 2441–2450.
18. Al-Hooti, M. H. A., Ahmad, T., & Djanali, S. (2019). Reduced difference expansion. *IAENG IJCS*, 46(4).
19. Tan, D., Lu, Y., Yan, X., & Wang, X. (2019). A simple review of audio steganography. *ITNEC*, 1409–1413.
20. Maniriho, P., & Ahmad, T. (2018). Improved data hiding via reduced difference expansion. *Engineering Letters*, 26(1).
21. Haque, Z., Saha, A., Das, T. S., Basu, A., & Chattopadhyay, A. (2018). Audio watermarking in EMD framework. *ICRCICN*, 269–274.
22. Sharma, A., & Kalra, P. K. (2016). Robust steganography using DWT and RSA. *Procedia CS*, 85, 490–500.
23. Chakrabarti, A., & Ghosal, A. (2014). Hybrid encryption using RSA and AES. *IJCA*, 92(5), 1–6.
24. Khalil, M., & Adib, A. (2014). Audio watermarking via frequency hopping. *ICMCS*, 211–216.
25. Kaur, M., & Sharma, N. (2013). Review of LSB steganography techniques. *IJARCSSE*, 3(4), 449–452.
26. Wang, H., Wang, S., & Wang, K. (2012). Image steganography with AES and Blowfish. *JIHMS*, 3(3), 221–233.
27. Cheddad, A., Condell, J., Curran, K., & McKevitt, P. (2010). Digital image steganography: Survey and analysis. *Signal Processing*, 90(3), 727–752.
28. Kekre, H. B., Athawale, A., & Patil, V. (2010). LSB matching revisited. *IJCA*, 6(9), 18–22.
29. Zaidan, B. B., Zaidan, A. A., Jalab, H. A., & Alanazi, H. O. (2010). Stego-image generator using cryptography. *IJCSNS*, 10(5), 28–35.
30. Mielikainen, J. (2006). LSB matching revisited. *IEEE SPL*, 13(5), 285–287.
31. Wang, Z., Bovik, A. C., Sheikh, H. R., & Simoncelli, E. P. (2004). Structural similarity image quality metric (SSIM). *IEEE TIP*, 13(4), 600–612.
32. Kharrazi, M., Sencar, H. T., & Memon, N. (2004). Concepts and practice of image steganography. *Wiley Encyclopedia of Telecommunications*.
33. Fridrich, J., Goljan, M., & Du, R. (2001). Detection of LSB steganography. *ACM Workshop*, 27–30.

34. Johnson, N. F., Duric, Z., & Jajodia, S. (2001). *Steganography and watermarking—Attacks and countermeasures*. Springer.
35. Malvar, H. S., & Florêncio, D. A. (2002). Spread spectrum for watermarking. *ICASSP*, IV-3301.
36. Cox, I. J., Kilian, J., Leighton, F., & Shamoon, T. (1995). Secure spread spectrum watermarking. *NEC Tech Report*, 95-10.
37. Petitcolas, F. A. P., Anderson, R. J., & Kuhn, M. G. (1999). Information hiding—A survey. *IEEE*, 87(7), 1062–1078.
38. Anderson, R. J., & Petitcolas, F. A. P. (1998). Limits of steganography. *IEEE JSAC*, 16(4), 474–481.
39. Provos, N., & Honeyman, P. (2003). Hide and seek: Steganography intro. *IEEE Security & Privacy*, 1(3), 32–44.
40. Rivest, R. L., Shamir, A., & Adleman, L. (1978). RSA algorithm. *Communications of the ACM*, 21(2), 120–126. <https://doi.org/10.1145/359340.359342>





11% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.




Filtered from the Report

- Bibliography
- Quoted Text

Match Groups

-  **38 Not Cited or Quoted 7%**
Matches with neither in-text citation nor quotation marks
-  **16 Missing Quotations 3%**
Matches that are still very similar to source material
-  **0 Missing Citation 0%**
Matches that have quotation marks, but no in-text citation
-  **0 Cited and Quoted 0%**
Matches with in-text citation present, but no quotation marks

Top Sources

- 7%  Internet sources
- 6%  Publications
- 9%  Submitted works (Student Papers)

Integrity Flags

0 Integrity Flags for Review

No suspicious text manipulations found.

Our system's algorithms look deeply at a document for any inconsistencies that would set it apart from a normal submission. If we notice something strange, we flag it for you to review.

A Flag is not necessarily an indicator of a problem. However, we'd recommend you focus your attention there for further review.