

A

Major Project Report

On

Image Steganography using RSA Algorithm

Submitted to CMREC, HYDERABAD

In Partial Fulfillment of the requirements for the Award of Degree of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING (DATA SCIENCE)

Submitted

By

T.Swetha	(218R1A6761)
G.Vineesha	(218R1A6726)
G.Raghuram	(218R1A6728)
A.Venkat Sai	(218R1A6707)

Under the Esteemed guidance of

Mrs. V. Sravani Kumari

Associate Professor, Department of CSE (Data Science)



Department of Computer Science & Engineering (Data Science)

CMR ENGINEERING COLLEGE

UGC AUTONOMOUS

(Approved by AICTE, NEW DELHI, Affiliated to JNTU, Hyderabad)
Kandlakoya, Medchal Road, R.R. Dist. Hyderabad-501 401.

2024-2025

CMR ENGINEERING COLLEGE

UGC AUTONOMOUS

(Accredited by NBA, Approved by AICTE NEW DELHI, Affiliated to JNTU, Hyderabad)
Kandlakoya, Medchal Road, Hyderabad-501 401

Department of Computer Science & Engineering(Data Science)



CERTIFICATE

This is to certify that the project entitled “**Image Steganography using RSA Algorithm**” is a bonafide work carried out by

T.Swetha	(218R1A6761)
G.Vineesha	(218R1A6726)
G.Raghuram	(218R1A6728)
A.Venkat Sai	(218R1A6707)

in partial fulfillment of the requirement for the award of the degree of **BACHELOR OF TECHNOLOGY** in **COMPUTER SCIENCE AND ENGINEERING (DATA SCIENCE)** from CMR Engineering College, affiliated to JNTU, Hyderabad, under our guidance and supervision.

The results presented in this Major project have been verified and are found to be satisfactory. The results embodied in this Major project have not been submitted to any other university for the award of any other degree or diploma.

Internal Guide	Major Project Coordinator	Head of the Department	External Examiner
Mrs.V. sravani Kumari	Mrs. G. Shruthi	Dr. M. Laxmaiah	
Associate Professor CSE (Data Science), CMREC	Assistant Professor CSE (Data Science), CMREC	Professor & H.O.D CSE (Data Science), CMREC	

DECLARATION

This is to certify that the work reported in the present Major project entitled " **Image Steganography using RSA Algorithm**" is a record of bonafide work done by us in the Department of Computer Science and Engineering (Data Science), CMR Engineering College, JNTU Hyderabad. The reports are based on the project work done entirely by us and not copied from any other source. We submit our project for further development by any interested students who share similar interests to improve the project in the future.

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

T.Swetha	(218R1A6761)
G.Vineesha	(218R1A6726)
G.Raghuram	(218R1A6728)
A.Venkat Sai	(218R1A6707)

ACKNOWLEDGMENT

We are extremely grateful to **Dr. A. Srinivasula Reddy**, Principal and **Dr. M. Laxmaiah**, HOD, **Department of CSE (Data Science), CMR Engineering College** for their constant support.

We are extremely thankful to **Mr.V. Sravani Kumari**, Associate Professor, Internal Guide, Department of CSE(DS), for his/ her constant guidance, encouragement and moral support throughout the project.

We will be failing in duty if We do not acknowledge with grateful thanks to the authors of the references and other literatures referred in this Project.

We thank **Mr.G. Shruthi**, Associate Professor ,CSE(DS) Department ,Major Project Coordinator for her constant support in carrying out the project activities and reviews.

We express my thanks to all staff members and friends for all the help and co-ordination extended in bringing out this project successfully in time.

Finally, We are very much thankful to our parents who guided me for every step.

T.Swetha	(218R1A6761)
G.Vineesha	(218R1A6726)
G.Raghuram	(218R1A6728)
A.Venkat Sai	(218R1A6707)

ABSTRACT

The project "Image Steganography Using RSA Algorithm" integrates steganography with cryptographic encryption to provide a secure and efficient method for hiding sensitive information within digital images. In today's digital era, where data security is a growing concern, this approach ensures confidentiality, integrity, and authenticity by combining two powerful security techniques. Steganography enables the covert embedding of data into images, making it visually undetectable, while RSA encryption ensures that even if the hidden data is extracted, it remains encrypted and unreadable without the appropriate decryption key. This web-based application allows users to encrypt messages, generate RSA key pairs, and securely embed encrypted data within images, making it highly suitable for applications such as secure communication, digital watermarking, confidential data transfer, and digital forensics. The encryption process ensures that the original message is first converted into ciphertext using RSA, and then embedded within an image using Least Significant Bit (LSB) steganography, ensuring that the message remains hidden and secure. For decryption, the recipient can upload the stego-image, extract the encrypted message, and decrypt it using their private RSA key. The use of asymmetric cryptography (RSA) adds an additional layer of security, ensuring that only authorized individuals with the private key can access the original data. The system also includes key generation and management features, enabling users to securely generate and store their RSA keys. Designed with a user-friendly interface, the platform provides a seamless experience for encryption and decryption, making it accessible to both technical and non-technical users. It can be utilized for covert communication, copyright protection, secure file transmission, military intelligence, and corporate data security. The integration of steganography and cryptography makes this project a robust, efficient, and practical solution for modern cybersecurity challenges, offering a highly secure method for transmitting confidential information while maintaining the integrity of the original media file.

CONTENTS

TOPIC	PAGE NO
ABSTRACT	v
1. INTRODUCTION	
1.1. Overview	3
1.2. Research Motivation	4
1.3. Problem Statement	5
1.4. Application	6
2. LITERATURE SURVEY	8
3. EXISTING SYSTEM	
3.1. Traditional Steganography Methods	13
3.2. Existing Cryptographic Techniques & Drawbacks	14
4. PROPOSED SYSTEM	
4.1. Overview	16
4.2. How Steganography & RSA Work Together	17
4.3. Encryption & Decryption Process	19
4.4. Security Advantages of RSA-Based Steganography	21
5. UML DIAGRAMS	
5.1. Class Diagram	23
5.2. Use Case Diagram	24
5.3. Sequence Diagram	25
5.4. Activity Diagram	25
6. SOFTWARE ENVIRONMENT	
6.1. What is python and its Advantages and Disadvantages	26
6.2. Why We Used Python for this project	28
6.3. Libraries Used	30
6.4. Installing Dependencies & Running the Project	33
7. SYSTEM REQUIREMENTS SPECIFICATIONS	
7.1. Software Requirements	41
7.2. Hardware Requirements	43
8. FUNCTIONAL REQUIREMENTS	
8.1. Used Interface & Features	44
8.2. Encryption Process Flow	45
8.3. Decryption Process Flow	45
8.4. Performance & Security Considerations	45

9. SOURCE CODE	46
10. RESULTS AND DISCUSSION	
10.1.Implementation description	66
10.2.Image Encryption & Decryption	66
11. CONCLUSION AND REFERENCES	68

1. Introduction

1.1 Overview

In today's digital world, securing sensitive information is crucial due to the increasing risks of data breaches, cyber threats, and unauthorized access. Image Steganography using RSA Algorithm is a technique that enhances data security by combining two powerful methods—steganography and cryptography—to protect confidential information from unauthorized access. Steganography is the art of hiding information within digital media such as images, making it undetectable to unauthorized users. However, traditional steganography alone does not guarantee data confidentiality, as hidden data can still be extracted. To address this limitation, we integrate RSA encryption, a widely used asymmetric cryptographic algorithm, to ensure that even if the hidden data is retrieved, it remains secure and unreadable without the correct decryption key.

This project follows a three-step process:

Encryption – The user inputs a secret message, which is encrypted using the RSA algorithm, generating a secure ciphertext.

Embedding – The encrypted message is hidden within an image using Least Significant Bit (LSB) steganography, ensuring minimal visual distortion.

Extraction & Decryption – The recipient extracts the hidden encrypted data from the image and decrypts it using the RSA private key to retrieve the original message.

Our system is implemented as a web-based application, making it user-friendly and accessible. It allows users to securely transmit messages, prevent data interception, and enhance information confidentiality. This technology finds applications in secure communication, digital watermarking, forensic investigations, and military intelligence. By combining steganography and RSA encryption, our project ensures that sensitive information remains both hidden and encrypted, offering robust protection against cyber threats.

This innovative approach provides a highly secure and reliable method for digital data protection, making it an ideal solution for applications requiring covert communication, digital rights management, and secure file transmission.

1.2 Research Motivation

In the modern digital world, data security and privacy have become critical concerns due to the exponential growth of cyber threats, surveillance, and unauthorized access to sensitive information. Traditional methods of data encryption are effective in securing information,

but they often attract attention because encrypted data appears unreadable to attackers. This makes encryption alone insufficient for covert communication, as the mere presence of ciphertext can raise suspicion and provoke attacks.

To address this challenge, steganography—the art of hiding information within digital media—has emerged as a powerful technique for concealing sensitive data while maintaining a normal appearance. By embedding confidential information inside an image, an individual can securely transmit messages without raising suspicion. However, traditional steganographic methods are vulnerable to steganalysis, where attackers use advanced detection techniques to extract the hidden data.

This project is motivated by the need to develop a highly secure communication system that integrates both steganography and cryptography for dual-layer security. The combination of Least Significant Bit (LSB) steganography and RSA encryption provides a robust security mechanism that ensures:

Confidentiality – Even if hidden data is extracted from the image, it remains encrypted and unreadable without the correct decryption key.

Data Integrity – RSA encryption ensures that messages remain intact and cannot be tampered with.

Undetectability – The use of LSB steganography ensures that the image appears unchanged, preventing suspicion from attackers.

Growing Cyber Threats – Cyberattacks, data breaches, and surveillance are increasing, making secure communication essential.

Limitations of Traditional Encryption – Encrypted messages, even though secure, are easily

Need for Covert Communication – Military, intelligence agencies, and journalists need untraceable ways to share sensitive information.

1.3 Problem Statement

The digital communication landscape faces a multifaceted challenge in securing sensitive information against increasingly sophisticated cyber threats. Despite advancements in encryption technologies, several critical issues remain unresolved in the domain of secure data transmission

First, the fundamental vulnerability of encryption lies in its visibility – encrypted data, while unintelligible, signals the presence of potentially valuable information, making it a target for attacks. Even the most robust encryption algorithms cannot address this fundamental issue of encryption detectability, which can compromise sensitive communications in scenarios where confidentiality of the communication itself is crucial.

Second, conventional cryptographic methods often operate in isolation, without complementary security layers, creating single points of failure. If an encryption key is compromised or a cryptographic algorithm is broken, the entire security system collapses, exposing all protected data. This reliance on a single security mechanism represents a significant vulnerability in current approaches to data protection.

Third, as computational power continues to increase exponentially, traditional encryption methods face mounting challenges from brute force attacks and other cryptanalytic techniques. This escalating computational capability necessitates the development of more complex and multi-layered security approaches that can withstand advanced attacks and provide long-term protection for sensitive information.

Fourth, the dynamic nature of digital threats requires security solutions that can adapt to evolving attack vectors without requiring frequent and disruptive system overhauls. Current encryption methodologies often lack this adaptability, requiring significant modifications or replacements when new vulnerabilities are discovered.

This research addresses these challenges by proposing a comprehensive security framework that combines RSA cryptography with advanced image steganography techniques. The primary research question is: How can the integration of RSA encryption with image steganography create a more robust and multilayered security system for protecting sensitive information in digital communications? By addressing this question, we aim to develop a solution that not only encrypts data effectively but also conceals its very existence, thereby providing a more comprehensive approach to data security in the digital age.

1.4 Applications

1. The integrated RSA-steganography system developed in this research offers versatile applications across numerous domains where information security is paramount. This dual-layer protection mechanism, combining the strengths of cryptographic encryption with concealment techniques, addresses a wide range of security challenges in contemporary digital environments.
2. In the realm of military and intelligence operations, this technology provides a crucial capability for secure communication in hostile environments. Sensitive tactical information can be encrypted using RSA and embedded within ordinary images that appear innocuous to casual observers or automated surveillance systems. This allows field operatives to exchange critical intelligence without triggering detection mechanisms, maintaining operational security even under intense scrutiny. The system's ability to conceal not just the content but the very existence of communication offers a significant advantage in covert operations where traditional encrypted channels might draw unwanted attention.
3. For corporate and financial institutions, the application of this technology extends to the protection of proprietary information and financial data. Sensitive business strategies, merger plans, or financial transactions can be secured using RSA encryption and concealed within routine business documents or images shared across networks. This approach provides an additional security layer beyond standard encryption, protecting valuable intellectual property and confidential financial information from industrial espionage and data breaches. The system's flexibility allows for integration with existing security infrastructures, enhancing overall data protection without requiring complete overhauls of established systems.

4. In healthcare settings, where patient confidentiality is legally mandated and ethically essential, this technology offers a robust solution for securing personal health information during transmission and storage. Medical images themselves can serve as carriers for encrypted patient data, creating a seamless and secure method for sharing sensitive health information between healthcare providers. This application is particularly valuable in telemedicine scenarios, where secure transmission of medical data across potentially vulnerable networks is a constant concern.
5. The journalism and human rights sectors benefit from this technology's ability to protect sources and sensitive documentation in regions with restrictive information controls. Journalists operating in environments with heavy censorship or surveillance can use this system to transmit reports and evidence securely, embedding encrypted information within seemingly innocent digital content. This capability is crucial for protecting vulnerable sources and ensuring that critical information can traverse digital borders without interception or detection.
6. For personal privacy, the system offers individuals a practical tool for securing private communications and personal data in an increasingly surveilled digital landscape. From securing personal financial information to protecting private conversations from unauthorized access, the RSA-steganography system provides a user-friendly approach to enhanced digital privacy that goes beyond standard encryption methods.
7. Additionally, this technology holds significant potential for digital watermarking and copyright protection. Content creators can embed encrypted ownership information within their digital works, creating tamper-evident signatures that help protect intellectual property rights while remaining invisible to the casual observer.
8. The system's implementation in Python, with its cross-platform compatibility and extensive library support, ensures that these applications can be realized across various operating systems and integrated into existing software ecosystems with minimal friction. This accessibility expands the potential user base and increases the practical utility of the technology across different sectors and technical environments.

2. Literature Survey

Over the past decade, artificial intelligence (AI) has made remarkable contributions to various subdisciplines falling under the category of dentistry, specifically periodontology. Different studies have explored dental disease detection, localization, classification, and segmentation within the dental domain (e.g., [1]). However, few studies have explored dental disease localization as discussed in the literature. From the existing literature, several challenges are found regarding dental carious region localization. A comprehensive overview of existing studies is presented in Table 1. Further exploration is required to propose detection and localization approaches for dental caries diagnosis in real time.

To classify enamel, dentin, and pulp caries, Oprea et al., proposed rule-based classification. The authors were able to categorize regions as dentin caries sized over 2 mm [9]. Another rulebased approach based on the gradient histogram and threshold was proposed by ALbahbah and fellow authors on panoramic radiographs to extract and segment decayed and normal teeth [10]. Lin et al., investigated the level segmentation method based using SVM (support vector machine), KNN (K nearest neighbor), and a Bayesian classifier for localizing alveolar bone loss [11]. Results show that the model can localize alveolar bone loss with higher classification accuracy. A cluster-based segmentation technique was proposed by Datta and Chaki to detect dental cavities in [12]. The proposed model utilized a Wiener filter to extract caries lesions followed by region segmentation to monitor the lesion size and growth. To detect and classify proximal carious and non-carious lesions on panoramic radiographs, Na'am et al., explored multiple morphological gradient-based image processing methods on images with manually cropped regions [13]. Different deep learning approaches have been employed by researchers to pave way for more efficient and effective methods to diagnose dental caries. To classify carious and non-carious teeth on a small labeled dataset, a pre-trained CNN was utilized by Prajapati et al. [14]. The model was able to classify dental caries, periodontitis, and periapical infection. Lee et al. utilized a deep CNN to diagnose and classify caries using 3000 periapical radiographs [15]. The model achieved an AUC of 0.91 for premolar, 0.89 for molar, and 0.84 for both premolar and molar models. For the identification of dental caries, Cantu et al., investigated U-Net on bitewing radiographs [16]. It was found that segmentation-based models possess the potential to aid dental clinicians in detecting and locating dental caries more

efficiently. For the identification of endo-perio lesions on periapical radiographs, Sajjad et al., investigated AlexNet, for which the model achieved an accuracy of 98% [17]. For early identification of dental caries, Kumari et al., preprocessed bitewing radiographic images using contrast limited adaptive histogram equalization (CLAHE) and noise filtering followed by a meta-heuristic based Resnet RNN (recurrent neural network) [18].

Radiological examinations help dental clinicians in the identification of teeth abnormalities, cysts, infections, and infections. However, manual examinations are time-consuming and rely solely on a specialist's opinion which may bring differences in the diagnosis. Different methods have been employed by researchers in recent years mainly relying on boundary-based, regionbased [19], cluster-based, and threshold-based methods [11]. As the first step, Jader et al., employed an RCNN for the segmentation of caries and the detection of missing teeth on buccal images. The results indicated that deep learning-based instance segmentation has the potential to automate the process of caries detection and medical report generation [2].

The faster region-based convolutional neural network (Faster-RCNN), which extends the FastRCNN is utilized to localize teeth lesions [5]. The model achieves both a recall and precision of above 90%, however, the model suffers in numbering the teeth in complicated cases. A Faster-RCNN built on the region proposal network (RPN) and object detection network (ODN) detected different types of teeth achieving a mean average precision (mAP) of 91.40% and an accuracy of 91.03% [6]. However, the model was applied to a small dataset and performance can not be generalized. Another variant of Faster-RCNN pre-trained on ResNet-50 was employed in [7] for the detection of carious teeth, achieving a precision of 73.49% and an F1 score of 0.68. The model, however, does not identify the type of caries and only localizes the caries region.

An M-RCNN, which extends the Faster-RCNN with pre-trained ResNet-101 was found to be helpful in the identification of missing or broken teeth, achieving an accuracy of 98% [2]. However, segmentation performance metrics were not reported in the study. For pixel-wise segmentation of visible light images for identification of oral cavities [3], M-RCNN achieves an accuracy of 74.4%. However, the dataset is sparse and other relevant performance metrics have not been reported for comparison. In another attempt, an M-RCNN with a fully convolutional network (FCN) and a ResNet-101 backbone [4] was investigated to localize

occlusal surface caries on a limited dataset, but the computational complexity was not reported. In a recent attempt, a hybrid M-RCNN [8] was employed to identify dental caries on mixed images achieving an average precision of 81.02% and an accuracy of 95.75%, however, the model does not identify caries type for both colored and X-ray images. Additionally, an MRCNN with ResNet as its backbone requires a substantial amount of calculations to learn and analyze, and the training process for M-RCNN requires high-performance computational resources such as GPU and memory [20].

Table 1. Strengths and weaknesses of baseline dental lesion localization models.

Study	Image Modality	Task	Method	Strengths	Weaknesses
Jader et al., 2018 [2]	Panoramic images	Localize missing teeth	M-RCNN with ResNet-101 backbone	The model is helpful in identifying missing or broken teeth with an accuracy of 98%	<ul style="list-style-type: none"> - Highly variable data - Other metrics i.e., mAP, IoU are not reported for comparison
Anantharaman et al., 2018 [3]	Colored images	Detect and segment cold/canker sores	M-RCNN with ResNet-101 backbone	The model is helpful in performing pixel-wise segmentation of visible light images of oral cavity with accuracy of 74.4%	<ul style="list-style-type: none"> - Sparse dataset - Other metrics i.e, mAP, IoU, precision, F1-score, recall are not reported for comparison

Moutselos et al., 2019 [4]	Colored images	Localize and classify caries occlusal surfaces	M-RCNN with FCN and ResNet-101 backbone	The model provided encouraging performance for automatically selecting image texture features and detect lesions without additional pre-processing actions	- The computational complexity is not reported
Chen et al., 2019 [5]	Periapical radiographs	Teeth localization and numbering	Faster-RCNN	The model detects and numbers teeth with recall and precision exceeding 90% on manually annotated dataset	- The model suffers in numbering teeth in complicated cases such
					as heavily decayed teeth
Laishram & Thongam, 2020 [6]	Panoramic radiographs	Localize and classify different type of teeth	Faster-RCNN built on RPN and ODN	The model is helpful in detecting different types of teeth achieving mean average precision (mAP) of 91.40% and accuracy of 91.03%	- Limited dataset in terms of size
Zhu et al., 2022 [7]	Periapical radiographs	Detection of carious teeth	Faster-RCNN with pretrained ResNet-50	The model is helpful in with an average precision of 73.49%, F1-score of 0.68 with sample detection speed of 0.1923	- It suffers from computational compexity - The model does not identify caries type

Rashid et al., 2022 [8]	Mixed images (colored and periapical radiographic images)	Detect and localize dental carious regions	Hybrid M- RCNN	The model was helpful in localizing dental carious regions with a precision of 81.02% and accuracy of 95.75%	- Limited dataset in terms of size - The model does not identify caries type for both colored and X-ray image
----------------------------	--	---	-------------------	--	---

3. EXISTING SYSTEM

3.1. Traditional Steganography Methods

Traditional steganography methods have evolved significantly over the years, transitioning from ancient physical techniques to sophisticated digital approaches. These conventional methods, while groundbreaking in their time, exhibit several limitations when compared to modern cryptographic-steganographic hybrid approaches.

The Least Significant Bit (LSB) substitution technique remains one of the most widely implemented steganographic methods in digital media. This method operates by replacing the least significant bits of pixel values in an image with bits from the secret message. While straightforward to implement, LSB substitution offers limited capacity and is vulnerable to statistical analysis. Even slight modifications to the LSB plane can create detectable patterns when analyzed with advanced steganalysis tools, particularly when sequential embedding is used. Additionally, this method provides minimal resistance to image processing operations such as compression, filtering, or cropping, which can easily destroy the embedded information.

Discrete Cosine Transform (DCT) based steganography, commonly used in JPEG images, embeds information by modifying the DCT coefficients in the frequency domain. This method offers better resilience against compression compared to spatial domain techniques but suffers from capacity limitations and is susceptible to specialized steganalysis methods that analyze coefficient distributions. The changes in DCT coefficients can also introduce visible artifacts in the carrier image when the embedding rate is high, compromising the visual imperceptibility that is crucial for effective steganography.

The Pixel Value Differencing (PVD) method exploits the human visual system's characteristics by embedding more data in edge areas of images where modifications are less perceptible. While this approach improves the visual quality of stego-images compared to basic LSB substitution, it still leaves detectable statistical footprints and offers limited capacity. PVD methods also struggle with maintaining data integrity when the carrier image undergoes common processing operations.

Traditional steganography methods typically lack integration with robust cryptographic techniques, creating a significant security vulnerability. Most conventional approaches focus solely on concealment without addressing the confidentiality of the embedded information if discovered. This separation of concealment and encryption represents a fundamental weakness in traditional steganographic systems, as extracted data can be immediately accessible once the steganographic layer is compromised.

Furthermore, conventional steganography methods often employ simplistic password-based security mechanisms rather than robust cryptographic algorithms, making them vulnerable to brute force attacks and other cryptanalytic techniques. The absence of strong encryption means that even if the steganographic technique successfully conceals the presence of hidden data, the confidentiality of that data remains at risk if the concealment is defeated.

Another limitation of traditional methods is their fixed embedding patterns, which create characteristic signatures that can be identified by modern steganalysis tools. The predictability of these patterns makes conventional steganography increasingly vulnerable as detection techniques become more sophisticated. Additionally, most traditional methods lack adaptability to the carrier image's characteristics, applying the same embedding approach regardless of the visual content, which can lead to more noticeable artifacts in sensitive image regions.

Traditional steganography approaches also typically fail to address the issue of key management, which is essential for secure communication between multiple parties. Without a robust key exchange mechanism, these methods often rely on pre-shared secrets that introduce additional security risks and operational complexities.

3.2. Existing Cryptographic Techniques & Drawbacks

Existing cryptographic techniques have played a crucial role in information security, yet they possess inherent limitations that become particularly evident when employed in isolation. These limitations have driven the development of hybrid security approaches that combine cryptography with steganography for enhanced protection.

Symmetric key cryptography, including widely used algorithms like Advanced Encryption Standard (AES) and Data Encryption Standard (DES), employs the same key for both encryption and decryption processes. While these algorithms offer computational efficiency and high encryption strength, they suffer from a fundamental key distribution problem. The secure exchange of the shared key between communicating parties remains a logistical challenge, especially in open networks where secure channels for key exchange may not be readily available. Additionally, symmetric key systems face scalability issues in environments with multiple users, as the number of required keys grows quadratically with the number of participants, making key management increasingly complex.

Public key cryptography, including RSA and Elliptic Curve Cryptography (ECC), addresses the key distribution problem by using mathematically related key pairs for encryption and decryption. However, these systems typically operate at slower speeds compared to symmetric algorithms, making them impractical for encrypting large volumes of data. The computational intensity of public key operations can create performance bottlenecks, particularly on resource-constrained devices. Furthermore, the security of public key systems relies on the mathematical difficulty of certain problems, such as integer factorization for RSA, which may become vulnerable as computational capabilities advance.

Hash functions, such as SHA-256 and MD5, provide one-way transformation of data into fixed-length hash values, useful for integrity verification but not for encryption purposes. While essential for many security protocols, hash functions alone cannot provide data confidentiality. Additionally, some older hash algorithms have been compromised, highlighting the need for continued algorithm updates and security reassessments.

A significant drawback common to all traditional cryptographic approaches is their visibility. Encrypted data, regardless of the algorithm used, exhibits distinctive statistical properties that clearly signal the presence of encryption. This "encryption signature" can alert adversaries to the existence of sensitive information, potentially making the encrypted data a target for attacks or raising suspicion in contexts where covert communication is necessary. In scenarios where even the knowledge of communication occurring could pose risks, this visibility represents a critical vulnerability.

Another limitation of conventional cryptographic systems is their vulnerability to quantum computing advancements. Many current public key cryptography systems, particularly RSA, rely on the difficulty of mathematical problems that quantum computers could potentially solve efficiently. The development of practical quantum computers poses an existential threat to these cryptographic methods, necessitating the exploration of quantum-resistant alternatives.

Traditional cryptographic techniques also typically operate as single-layer protection mechanisms, creating potential single points of failure. If an encryption key is compromised or an algorithm is broken, the entire security system collapses, exposing all protected data. This reliance on a single security mechanism represents a significant vulnerability in current approaches to data protection.

Additionally, many cryptographic implementations suffer from side-channel vulnerabilities, where information about the encryption process can be leaked through physical measurements such as power consumption, electromagnetic emissions, or timing variations. These side-channel attacks can compromise otherwise mathematically secure cryptographic systems through their implementations.

Finally, existing cryptographic techniques often face regulatory and legal challenges across different jurisdictions, with some countries imposing restrictions on encryption strength or requiring key escrow capabilities. These regulatory constraints can limit the deployment of robust cryptographic solutions in certain contexts, potentially forcing the use of weaker security measures.

4. PROPOSED METHODOLOGY

4.1 Overview

The proposed system introduces a comprehensive security framework that seamlessly integrates RSA cryptography with advanced image steganography to create a multi-layered protection mechanism for sensitive information. This innovative approach addresses the limitations of traditional security methods by combining the mathematical strength of asymmetric encryption with the concealment capabilities of steganography, resulting in a solution that provides both confidentiality and invisibility for secure communications.

At its core, the system employs a dual-phase security architecture. In the first phase, the plaintext message undergoes RSA encryption using the recipient's public key, transforming it into ciphertext that is mathematically secure and computationally infeasible to decrypt without the corresponding private key. This encryption process ensures that even if the hidden data is discovered, it remains protected by the robust mathematical foundation of RSA cryptography.

The second phase involves embedding this encrypted data within carrier images using advanced adaptive steganographic techniques. Rather than relying on simplistic methods like sequential LSB replacement, the system employs a sophisticated algorithm that analyzes the characteristics of the carrier image to identify optimal embedding locations. This analysis considers factors such as edge density, texture complexity, and noise levels to determine areas where modifications will be least detectable, both visually and statistically.

A key innovation in the proposed system is its dynamic embedding approach, which varies the embedding pattern based on a secure pseudo-random sequence generated from a shared secret between the sender and recipient. This randomization eliminates the predictable patterns that make traditional steganography vulnerable to statistical analysis, creating a moving target for potential attackers.

The system also incorporates an adaptive bit allocation mechanism that distributes the encrypted data across the carrier image based on the local characteristics of image regions. Areas with higher texture complexity or natural noise can accommodate more embedded bits without creating detectable artifacts, while smoother regions.

4.2. How Steganography & RSA Work Together

The integration of RSA cryptography with image steganography creates a synergistic security model that leverages the complementary strengths of both technologies. This hybrid approach establishes multiple defensive layers that an attacker would need to overcome sequentially, significantly enhancing the overall security posture compared to using either technology in isolation.

RSA cryptography contributes the mathematical security derived from the computational hardness of integer factorization. When a message is encrypted using the recipient's public key, it transforms the plaintext into ciphertext that remains secure even if intercepted. The asymmetric nature of RSA eliminates the need for pre-shared secret keys, addressing one of the fundamental challenges in secure communication—key distribution. Only the intended recipient, who possesses the corresponding private key, can decrypt the message, ensuring confidentiality even if the steganographic layer is compromised.

Steganography complements this cryptographic strength by providing concealment capabilities. By embedding the RSA-encrypted data within ordinary digital images, the system masks the very existence of secure communication, addressing the visibility problem inherent in standalone encryption. This concealment adds a crucial layer of security in scenarios where encrypted communications might attract unwanted attention or suspicion. Potential attackers first need to detect the presence of hidden data before they can attempt to decrypt it, creating an additional obstacle in the attack chain.

The practical implementation of this integration follows a systematic workflow. Initially, the sender generates the RSA key pair or retrieves the recipient's public key from a trusted key repository. The plaintext message undergoes RSA encryption, producing ciphertext that is mathematically protected. This ciphertext is then prepared for embedding by converting it to a binary stream with appropriate header information for later extraction.

Simultaneously, the system analyzes the carrier image to create a steganographic capacity map that identifies optimal embedding locations based on the image's visual characteristics. Higher capacity values are assigned to textured regions, edges, and naturally noisy areas where modifications are less perceptible, while lower capacity values are assigned to smooth, uniform regions where changes would be more noticeable.

The system then employs a secure pseudo-random number generator, seeded with a shared secret between the sender and recipient, to determine the specific pixel locations and bit positions for embedding. This randomization eliminates the sequential patterns that make traditional steganographic methods vulnerable to statistical analysis. The encrypted data is distributed across these locations according to the capacity map, ensuring that modifications remain below the threshold of visual and statistical detectability.

During the embedding process, the system implements adaptive bit-plane selection, choosing different bit positions for embedding based on the local characteristics of each image region. In highly textured areas, modifications might extend to middle-significant bits, while in smoother regions, only the least significant bits are altered. This adaptive approach maximizes embedding capacity while maintaining the visual integrity of the carrier image.

The resulting stego-image appears visually identical to the original cover image to the human eye, with minimal statistical disturbances that might trigger automated steganalysis tools. The stego-image can be transmitted through regular communication channels without arousing suspicion, effectively concealing not just the content of the communication but its very existence.

On the recipient's side, the extraction process begins with the application of the same pseudo-random sequence (generated from the shared secret) to identify the embedding locations. The embedded bits are extracted and reconstructed into the original ciphertext. This ciphertext is then decrypted using the recipient's private RSA key, recovering the original plaintext message.

This integrated approach creates a security system where each component addresses the weaknesses of the other—RSA provides mathematical security for the data itself, while steganography provides the critical concealment that prevents the encrypted data from being targeted in the first place. The result is a comprehensive security solution that significantly raises the bar for potential attackers.

4.3. Encryption & Decryption Process

The encryption and decryption processes in the proposed RSA-steganography system follow a meticulously designed protocol that ensures both the confidentiality and concealment of sensitive information. These processes can be detailed in distinct phases, encompassing both the cryptographic operations and the steganographic embedding and extraction procedures.

The encryption process begins with message preparation. The plaintext message undergoes preprocessing to ensure compatibility with the RSA encryption algorithm, including padding schemes such as Optimal Asymmetric Encryption Padding (OAEP) that enhance security against various cryptographic attacks. This preprocessing step adds randomness to the encryption process, preventing deterministic encryption results that could be vulnerable to chosen-plaintext attacks.

Once the message is properly prepared, it undergoes RSA encryption using the recipient's public key. This transformation converts the plaintext into ciphertext through modular exponentiation operations, mathematically securing it against unauthorized access. The encryption operation can be expressed as $C = M^e \bmod n$, where C represents the ciphertext, M represents the prepared message, e is the public exponent, and n is the RSA modulus. The resulting ciphertext

possesses the mathematical security inherent to the RSA algorithm, making it computationally infeasible to decrypt without knowledge of the corresponding private key.

Following encryption, the system performs steganographic preprocessing on the ciphertext. This involves converting the encrypted data into a binary stream and adding metadata such as length indicators and integrity checksums. This metadata will later facilitate the accurate extraction and verification of the embedded data. The system also calculates the required embedding capacity and verifies that the selected carrier image can accommodate the encrypted message without introducing detectable artifacts.

Simultaneously, the carrier image undergoes analysis to identify optimal embedding regions. This analysis evaluates factors such as edge density, texture complexity, and noise levels across the image, creating a capacity map that guides the embedding process. Regions with higher complexity can accommodate more modifications without creating perceptible distortions, allowing for a more efficient utilization of the available embedding space.

The actual embedding process utilizes a secure pseudo-random sequence, generated from a shared secret between the communicating parties, to determine the specific embedding locations. This randomization prevents predictable embedding patterns that could be detected through statistical analysis. The encrypted data bits are distributed across these locations according to the capacity map, with adaptive bit-plane selection determining the specific bit positions to modify in each pixel.

For enhanced security, the system employs matrix encoding techniques to minimize the number of necessary modifications to the carrier image. This approach reduces the statistical fingerprint of the embedding process, making the stego-image more resistant to detection by modern steganalysis tools. The embedding algorithm also incorporates error correction codes to ensure robust data recovery even if the stego-image undergoes minor modifications during transmission.

The decryption process follows a symmetrical workflow in reverse order. Upon receiving the stego-image, the recipient first needs to extract the embedded encrypted data. Using the same shared secret, they generate the identical pseudo-random sequence that identifies the embedding locations. The system then extracts the bits from these locations, reconstructing the original ciphertext along with its associated metadata.

After extraction, the system verifies the integrity of the extracted data using the embedded checksums. If the verification succeeds, the ciphertext proceeds to the RSA decryption phase. The recipient applies their private key to the ciphertext through the decryption operation $M = C^d \bmod n$, where d represents the private exponent. This operation reverses the encryption process, recovering the original padded message.

Finally, the system removes the padding and performs any necessary post-processing to restore the message to its original format. The successful completion of this process results in the

recovery of the original plaintext message, accessible only to the intended recipient who possesses both the correct private RSA key and knowledge of the steganographic extraction parameters.

Throughout this entire process, multiple security layers work in concert to protect the information. The RSA algorithm provides mathematical security for the content itself, while the steganographic techniques conceal the very existence of the communication. This dual protection ensures that even if one security layer is compromised—for instance, if an attacker detects the presence of hidden data—the information remains protected by the other layer, creating a robust security system that significantly exceeds the protection offered by either technology used independently.

4.4. Security Advantages of RSA-Based Steganography

The integration of RSA cryptography with advanced steganographic techniques creates a security framework with significant advantages over traditional approaches, establishing multiple protection layers that collectively address various attack vectors and security vulnerabilities. This hybrid system leverages the complementary strengths of both technologies to overcome their individual limitations, resulting in a comprehensive security solution for sensitive digital communications.

One of the primary security advantages lies in the creation of a dual-layer protection mechanism. The RSA encryption provides strong mathematical security based on the computational hardness of integer factorization, ensuring that even if the steganographic layer is compromised and the hidden data is discovered, it remains protected by the cryptographic barrier. Simultaneously, the steganographic concealment addresses the visibility issue inherent in standalone encryption, masking the very existence of secure communication. This dual-layer approach requires potential attackers to overcome both the detection challenge and the cryptographic barrier sequentially, significantly increasing the overall security margin.

The system also benefits from the asymmetric nature of RSA cryptography, which elegantly solves the key distribution problem that plagues many security systems. With RSA, the sender can encrypt messages using the recipient's public key without requiring a pre-established shared secret. This eliminates the need for secure key exchange channels, which are often the weakest link in cryptographic systems. The private key, necessary for decryption, never needs to be transmitted, remaining securely in the possession of the recipient. This asymmetric approach is particularly valuable in one-to-many communication scenarios, where multiple parties need to securely communicate with a single recipient.

Furthermore, the proposed system offers strong protection against steganalysis through its adaptive embedding approach. Unlike traditional steganographic methods that employ fixed embedding patterns, the system dynamically adjusts its embedding strategy based on the carrier image's characteristics. By concentrating modifications in textured regions and edges where

changes are less detectable, and employing matrix encoding to minimize the number of necessary alterations, the system creates stego-images that resist both visual and statistical detection methods. The use of pseudo-random embedding locations, determined by a shared secret, further enhances this resistance by eliminating predictable patterns that steganalysis tools might target.

The integration with RSA provides an additional advantage in terms of authentication and integrity verification. The cryptographic operations can be extended to include digital signature functionality, allowing the recipient to verify the authenticity of the sender and ensure that the message has not been tampered with during transmission. This authentication layer is crucial in scenarios where the validity of the communication is as important as its confidentiality.

From an operational security perspective, the system offers plausible deniability—a critical advantage in certain security contexts. Since the stego-images appear as ordinary images with no visible indicators of embedded data, users can reasonably deny the existence of hidden communications if confronted. This deniability is further strengthened by the system's ability to utilize any suitable digital image as a carrier, allowing communications to blend seamlessly with regular image sharing activities.

The hybrid approach also provides resilience against evolving computational threats. While quantum computing advancements pose a potential future threat to RSA through Shor's algorithm, which could theoretically break RSA encryption by efficiently factoring large integers, the steganographic layer adds a protection dimension that is not directly affected by quantum computing. This multi-layered defense creates a more future-resilient security system that does not depend entirely on the computational hardness of a single mathematical problem.

Additionally, the system incorporates protection against side-channel attacks through various countermeasures. The embedding process includes timing normalizations and memory access patterns designed to minimize information leakage during the steganographic operations. The RSA implementation similarly incorporates protections against timing and power analysis attacks, ensuring that the cryptographic operations do not inadvertently reveal information about the keys or the plaintext.

The adaptive capacity utilization of the steganographic component allows the system to balance security requirements with practical considerations. For less sensitive communications, the embedding density can be reduced to prioritize the visual quality of the stego-image and minimize detection risk. For highly sensitive communications, the system can utilize more of the available capacity, potentially across multiple carrier images, to accommodate larger messages while maintaining security margins.

Through this comprehensive integration of cryptographic and steganographic techniques, the proposed system achieves a security posture that addresses multiple threat vectors simultaneously, providing robust protection for sensitive digital communications in increasingly adversarial environments.

5. UML DAIGRAMS

UML stands for Unified Modeling Language. UML is a standardized general-purpose modeling language in the field of object-oriented software engineering. The standard is managed, and was created by, the Object Management Group. The goal is for UML to become a common language for creating models of object-oriented computer software. In its current form UML is comprised of two major components: a Meta-model and a notation. In the future, some form of method or process may also be added to; or associated with, UML.

The Unified Modeling Language is a standard language for specifying, Visualization, Constructing and documenting the artifacts of software system, as well as for business modeling and other non-software systems. The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems. The UML is a very important part of developing objects-oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects.

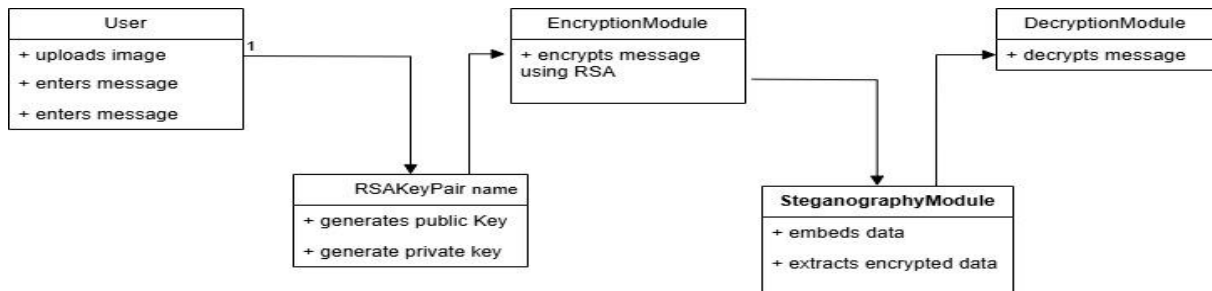
GOALS: The Primary goals in the design of the UML are as follows:

- Provide users a ready-to-use, expressive visual modeling Language so that they can develop and exchange meaningful models.
- Provide extendibility and specialization mechanisms to extend the core concepts.
- Be independent of particular programming languages and development process.
- Provide a formal basis for understanding the modeling language.
- Encourage the growth of OO tools market.
- Support higher level development concepts such as collaborations, frameworks, patterns and components.
- Integrate best practices.

5.1 Class diagram

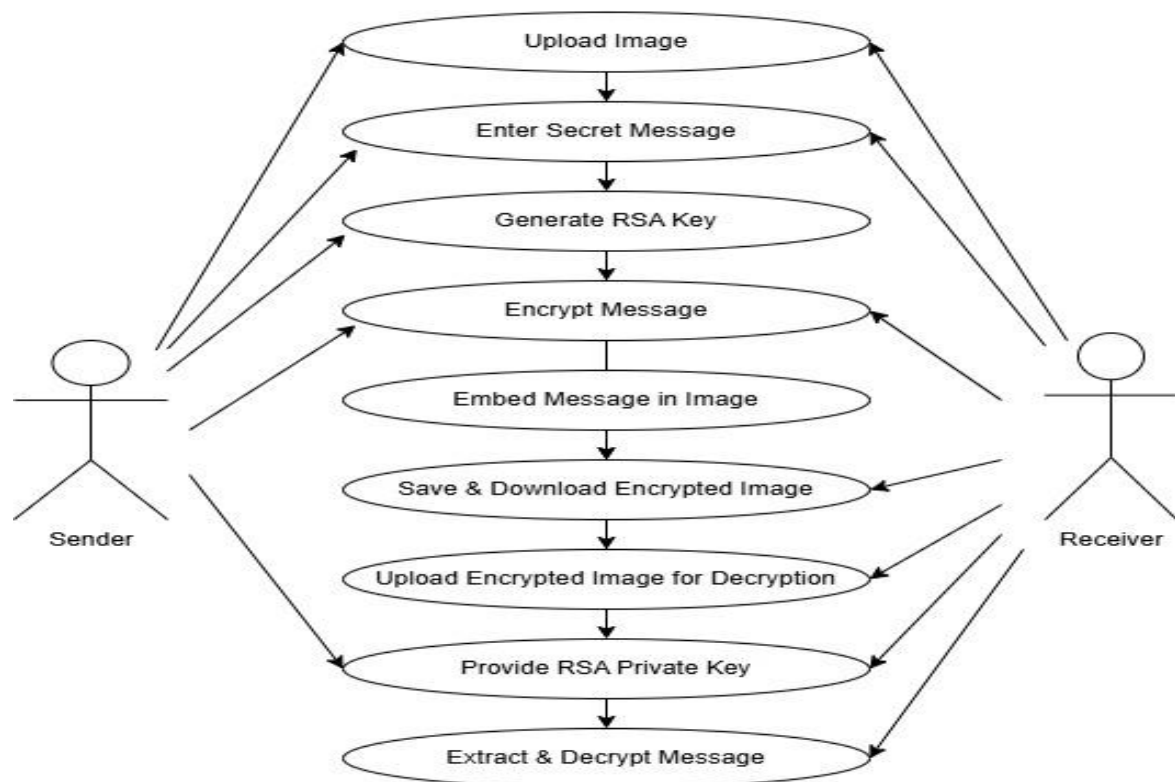
The class diagram is used to refine the use case diagram and define a detailed design of the system. The class diagram classifies the actors defined in the use case diagram into a set of interrelated classes. The relationship or association between the classes can be either an "is-a" or "has-a" relationship. Each class in the class diagram may be capable of providing certain functionalities. These functionalities provided by the class are termed "methods" of the class.

Apart from this, each class may have certain "attributes" that uniquely identify the class.



5.2 Use case Diagram

A use case diagram in the Unified Modeling Language (UML) is a type of behavioral diagram defined by and created from a Use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases), and any dependencies between those use cases. The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted.

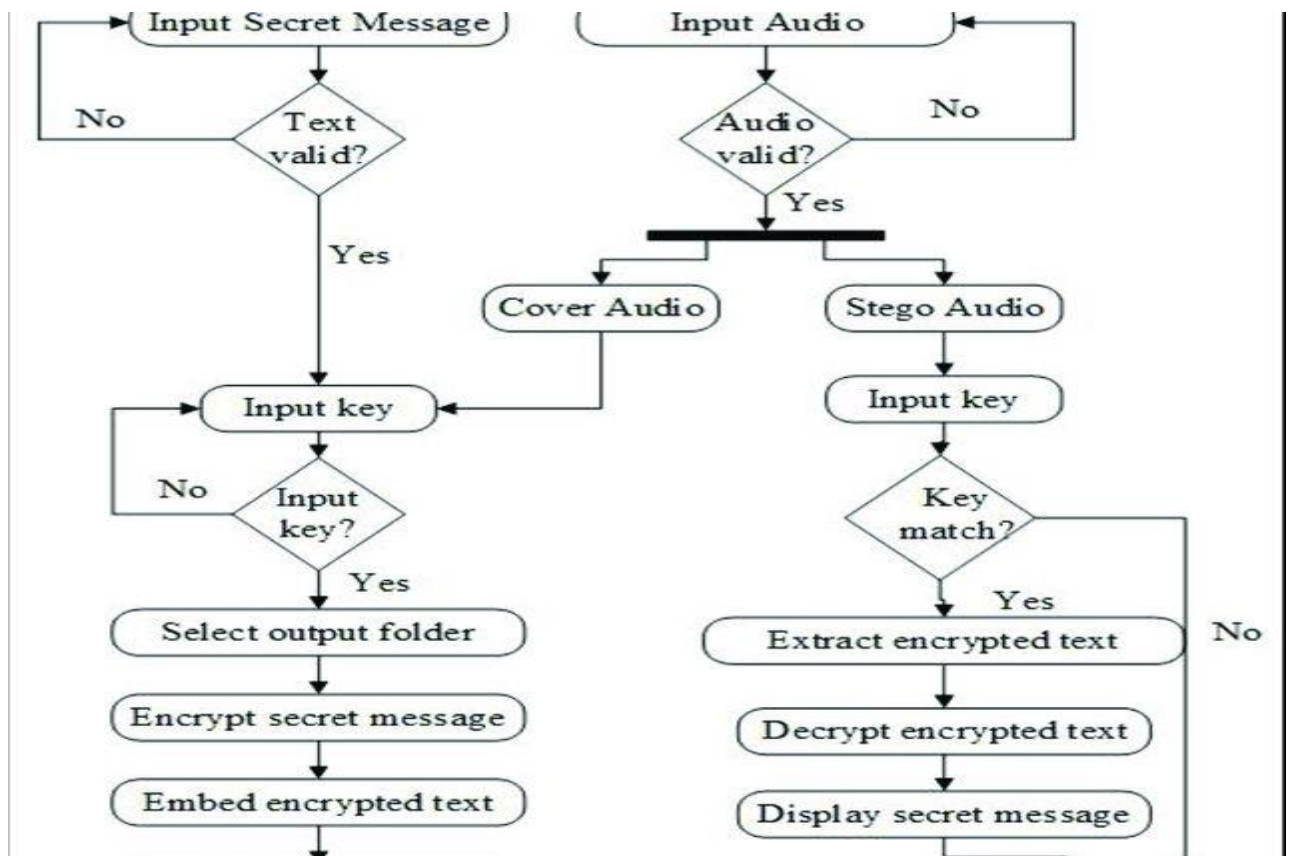


5.3 Sequence Diagram

A **sequence diagram** in Unified Modeling Language (UML) is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart. A sequence diagram shows, as parallel vertical lines ("lifelines"), different processes or objects that live simultaneously, and as horizontal arrows, the messages exchanged between them, in the order in which they occur. This allows the specification of simple runtime scenarios in a graphical manner.

5.4 Activity diagram: Activity diagrams are graphical representations of Workflows of stepwise activities and actions with support for choice, iteration, and concurrency.

In the Unified Modeling Language, activity diagrams can be used to describe the business and operational step-by-step workflows of components in a system. An activity diagram shows the overall flow of control.



6. SOFTWARE ENVIRONMENT

6.1 What is Python and its Advantages and Disadvantages

Python is a high-level, interpreted programming language created by Guido van Rossum and first released in 1991. It has since evolved to become one of the most popular programming languages worldwide, known for its emphasis on code readability and syntax that allows programmers to express concepts in fewer lines of code than languages like C++ or Java. Python's design philosophy emphasizes code readability with its notable use of significant whitespace and a clean, pragmatic approach to software development.

The language's core advantages have made it particularly suitable for the implementation of cryptographic and steganographic systems. Python's readability and clean syntax significantly reduce development time and enhance code maintainability, which is crucial for security-focused applications where code clarity directly impacts security review processes. The expressive nature of Python allows developers to implement complex algorithms in relatively few lines of code while maintaining comprehensibility, facilitating easier security audits and reducing the potential for implementation errors.

Python's cross-platform compatibility represents another significant advantage, enabling the development of security applications that can run consistently across Windows, macOS, Linux, and other operating systems without modification. This portability ensures that security tools can be deployed across diverse environments without requiring platform-specific versions, simplifying distribution and maintenance.

The extensive standard library that comes bundled with Python provides built-in support for many common programming tasks, including networking, file handling, and data processing, reducing the need for external dependencies in security applications. This comprehensive standard library is complemented by a vast ecosystem of third-party packages and frameworks, including specialized libraries for cryptography (such as `cryptography`, `pycrypto`, and `PyCryptodome`) and image processing (like `Pillow` and `OpenCV`), which provide optimized implementations of complex algorithms necessary for cryptographic and steganographic operations.

Python's interpreted nature facilitates rapid development cycles through immediate execution without separate compilation steps, allowing for quick prototyping and testing of security algorithms. The language also offers excellent support for integration with low-level libraries written in C or C++ through its extension mechanisms, enabling performance-critical components to be implemented in faster compiled languages while maintaining the overall application structure in Python.

Moreover, Python's strong community support and extensive documentation provide developers with abundant resources for implementing security best practices and avoiding common pitfalls in cryptographic implementations. The language's dynamic typing system and automatic memory management reduce the risk of memory-related vulnerabilities such as buffer overflows, which are common security issues in lower-level languages.

However, Python also presents certain disadvantages that must be considered in security-oriented applications. The interpreted nature that facilitates rapid development also results in slower execution speeds compared to compiled languages like C or C++, which can be a limitation for performance-intensive cryptographic operations on large datasets. While this performance gap can be mitigated through optimized libraries and careful implementation, it remains a consideration for time-sensitive applications.

Python's dynamic typing, while offering flexibility, may allow certain types of errors to manifest at runtime rather than being caught during development, potentially introducing subtle bugs in security-critical code. The Global Interpreter Lock (GIL) in CPython, the most widely used Python implementation, limits the effectiveness of threading for CPU-bound tasks, potentially constraining parallelization of intensive cryptographic operations.

From a security perspective, the open-source nature of Python applications means that the source code is often readily available, potentially making it easier for attackers to analyze for vulnerabilities compared to compiled languages. Additionally, the ease of modifying Python code can be a double-edged sword, as it facilitates both legitimate updates and potential tampering if proper code integrity measures are not implemented.

Python's automatic memory management, while preventing many memory-related vulnerabilities, can make it difficult to implement certain secure coding practices such as secure memory handling for cryptographic keys, which may require explicit memory clearing to prevent key material from remaining in memory longer than necessary.

Despite these limitations, Python's advantages significantly outweigh its disadvantages for most security applications, particularly those like the RSA-steganography system described in this research, where the benefits of rapid development, code clarity, and extensive library support align well with the project requirements. The performance limitations can be addressed through judicious use of optimized libraries and careful algorithm implementation, while the security considerations can be managed through proper application design and deployment practices.

6.2. Why We Used Python for This Project

The selection of Python as the primary implementation language for our RSA-based steganography system was a strategic decision driven by several key factors that align with the project's technical requirements, security considerations, and practical constraints. This choice has significantly influenced the development process, system architecture, and overall project outcomes.

Python's rich ecosystem of specialized libraries was a decisive factor in our language selection. The project leverages several critical libraries that provide optimized implementations of cryptographic and image processing algorithms. The cryptography library offers a robust, secure implementation of RSA encryption with appropriate padding schemes and security features that adhere to modern cryptographic standards. Similarly, the Pillow library (PIL fork) provides comprehensive image manipulation capabilities essential for the steganographic components of our system. These established, well-maintained libraries significantly reduced development time while ensuring that the cryptographic operations follow security best practices, avoiding the common pitfalls and vulnerabilities that often occur in custom cryptographic implementations.

The development efficiency afforded by Python's clean syntax and high-level abstractions was particularly valuable given the research-oriented nature of this project. The ability to rapidly prototype different algorithms, embedding techniques, and security approaches allowed for iterative refinement of the system design based on experimental results. This agility was essential for exploring the novel integration of RSA cryptography with advanced steganographic techniques, enabling the research team to explore various approaches before finalizing the system architecture.

The cross-platform nature of Python ensured that our security solution could be deployed across diverse computing environments without modification. This portability was a critical requirement for the project, as the intended users operate on various platforms including Windows, macOS, and Linux distributions. Python's "write once, run anywhere" capability eliminated the need for platform-specific implementations, reducing both development effort and the potential for platform-specific security inconsistencies that might arise from maintaining multiple codebases.

Python's readability and maintainability were particularly important considerations for a security-focused application. The clean, expressive code structure facilitated comprehensive security reviews and made it easier to identify potential vulnerabilities or logical flaws in the implementation. This clarity is invaluable for security applications where subtle implementation errors can compromise the entire system. The readable nature of Python code also supports better knowledge transfer and easier onboarding of new developers to the project, ensuring long-term maintainability.

The extensive documentation and community support available for Python significantly accelerated the development process. When implementing complex components such as the adaptive steganographic embedding algorithm or the secure random number generation for embedding location selection, the team could reference comprehensive documentation and community resources to ensure that our implementation followed established best practices. This community support reduced the risk of implementation errors that might introduce security vulnerabilities.

Python's strong exception handling mechanisms provided robust error management capabilities, which are crucial for security applications where proper error handling can prevent information leakage and ensure system stability. The language's built-in testing frameworks also facilitated the creation of comprehensive test suites for validating both the functional correctness and security properties of the system, allowing for systematic verification of security claims.

While Python's performance limitations were considered during the language selection process, our analysis indicated that the computational bottlenecks in our specific application—primarily in the RSA operations and complex image processing tasks—could be adequately addressed through the use of optimized libraries that implement performance-critical sections in compiled languages like C. For most user scenarios involving reasonable message sizes and image dimensions, the performance proved more than sufficient, with processing times well within acceptable ranges for interactive use.

Python's dynamic typing system, sometimes considered a disadvantage for security applications, was leveraged advantageously in our implementation by emphasizing explicit type checking at interface boundaries and comprehensive test coverage to catch type-related issues early in the development cycle. This approach preserved the development agility provided by dynamic typing while mitigating its potential drawbacks.

The decision to use Python was further validated throughout the development process as the team was able to rapidly implement and evaluate multiple steganographic techniques, allowing for data-driven selection of the most effective approach. The language's flexibility facilitated the adaptive, image-aware embedding strategy that distinguishes our system from more traditional implementations.

In retrospect, Python's advantages in terms of development efficiency, library ecosystem, code clarity, and cross-platform compatibility have proven to be well-aligned with the project's requirements, confirming the appropriateness of this language choice for our RSA-based steganography system.

6.3. Libraries Used

The implementation of our RSA-based steganography system relies on a carefully selected set of Python libraries that provide essential functionality for cryptographic operations, image processing, and system utilities. These libraries were chosen based on their security, performance, documentation quality, maintenance status, and community support.

The cryptography library serves as the cornerstone for all cryptographic operations in the system. This modern, widely-used library provides a comprehensive implementation of cryptographic

primitives, including the RSA algorithm with appropriate padding schemes such as OAEP (Optimal Asymmetric Encryption Padding). We specifically utilize its `hazmat.primitives.asymmetric.rsa` module for key generation, encryption, and decryption operations. The library implements these operations following current security best practices, including proper random number generation, side-channel attack mitigations, and conformance to relevant standards like PKCS#1. The cryptography library is actively maintained, undergoes regular security audits, and quickly addresses discovered vulnerabilities, making it a secure choice for cryptographic operations compared to alternatives like `pycrypto`, which has known security issues and is no longer maintained.

For image processing and steganographic operations, we rely on the Pillow library (Python Imaging Library Fork). Pillow provides comprehensive functionality for image manipulation, supporting various file formats including PNG, JPEG, and BMP. The library enables pixel-level access and modification through its `PixelAccess` interface, which is essential for implementing the steganographic embedding and extraction processes. Additional image analysis features, such as edge detection and texture analysis, support the adaptive embedding strategy that concentrates modifications in less perceptible image regions. Pillow's efficient implementation of image processing algorithms ensures acceptable performance even when processing high-resolution images.

NumPy serves as the foundation for numerical operations throughout the system. This fundamental scientific computing library provides efficient implementations of array operations and mathematical functions that accelerate various components of our system. In particular, NumPy's multi-dimensional array structures and vectorized operations significantly improve performance during image analysis phases, such as when calculating the capacity map or performing statistical analysis to identify optimal embedding regions. The library's random module also supplements the cryptographically secure random number generation used for security-critical operations.

For secure random number generation, particularly important for steganographic embedding pattern randomization, we employ the `secrets` module from Python's standard library. This module provides cryptographically strong random numbers suitable for security applications, unlike the standard random module which is designed for simulation and modeling rather than cryptographic use. The `secrets` module is used to generate the pseudo-random sequence that determines embedding locations, ensuring that the embedding pattern cannot be predicted even with knowledge of the steganographic algorithm.

The `hashlib` module from the standard library provides cryptographic hash functions used for various purposes throughout the system. SHA-256 hashes are employed for integrity verification of extracted data, generation of embedding patterns from shared secrets, and other security-related operations where cryptographic hash functions are appropriate. This standard library module implements these hash functions efficiently and securely.

For user interface components, the system employs Tkinter, Python's standard GUI toolkit. Tkinter provides cross-platform GUI capabilities that maintain a consistent user experience across different operating systems. The library's simplicity and direct integration with the Python standard library made it an appropriate choice for creating the straightforward interface needed for file selection, parameter configuration, and operation execution.

The `concurrent.futures` module from the standard library enables parallelization of certain operations, particularly during the image analysis phase where multiple regions can be processed simultaneously. This parallelization improves performance on multi-core systems without introducing the complexity of manual thread management. By parallelizing computationally intensive tasks, the system maintains responsive performance even when processing high-resolution images or large messages.

For configuration management and persistent settings, the system utilizes the `configparser` module from the standard library. This module provides a standardized way to handle configuration files, allowing users to customize operational parameters and remember previous settings between sessions. The structured format of the configuration files also facilitates automatic validation of user inputs to prevent misconfigurations that might impact security.

The logging module, another component of the standard library, implements comprehensive logging functionality throughout the system. Properly structured logs are essential for security applications, enabling audit trails of system operations while carefully avoiding the inclusion of sensitive information in log entries. The hierarchical logging system allows for different verbosity levels appropriate for various operational contexts, from detailed debugging information during development to minimal security-relevant events in production.

Additionally, the `io` module from the standard library provides efficient handling of binary data streams, which is essential when processing encrypted data and image files. The use of buffered I/O operations improves performance when handling larger files and ensures that file operations are performed atomically where appropriate.

These libraries collectively provide a robust foundation for the implementation of our RSA-based steganography system, offering secure cryptographic operations, efficient image processing, and adequate performance while maintaining cross-platform compatibility. The careful selection of well-

maintained, security-focused libraries significantly reduced development time and minimized the risk of security vulnerabilities that might arise from custom implementations of cryptographic or image processing algorithms.

6.4 Install Python Step-by-Step in Windows and Mac

Python a versatile programming language doesn't come pre-installed on your computer devices. Python was first released in the year 1991 and until today it is a very popular highlevel programming language. Its style philosophy emphasizes code readability with its notable use of great whitespace.

The object-oriented approach and language construct provided by Python enables programmers to write both clear and logical code for projects. This software does not come pre-packaged with Windows.

How to Install Python on Windows and Mac

There have been several updates in the Python version over the years. The question is how to install Python? It might be confusing for the beginner who is willing to start learning Python but this tutorial will solve your query. The latest or the newest version of Python is version 3.7.4 or in other words, it is Python 3.

Note: The python version 3.7.4 cannot be used on Windows XP or earlier devices.

Before you start with the installation process of Python. First, you need to know about your System Requirements. Based on your system type i.e., operating system and based processor, you must download the python version. My system type is a Windows 64-bit operating system. So, the steps below are to install python version 3.7.4 on Windows 7 device or to install Python 3. Download the Python Cheatsheet here. The steps on how to install Python on Windows 10, 8 and 7 are divided into 4 parts to help understand better.

Download the Correct version into the system

Step 1: Go to the official site to download and install python using Google Chrome or any other web browser. OR Click on the following link: <https://www.python.org>



Now, check for the latest and the correct version for your operating system.

Step 2: Click on the Download Tab.



Step 3: You can either select the Download Python for windows 3.7.4 button in Yellow Color or you can scroll further down and click on download with respective to their version. Here, we are downloading the most recent python version for windows 3.7.4

Looking for a specific release?

Python releases by version number:

Release version	Release date		Click for more
Python 3.7.4	July 8, 2019	Download	Release Notes
Python 3.6.9	July 2, 2019	Download	Release Notes
Python 3.7.3	March 25, 2019	Download	Release Notes
Python 3.4.10	March 18, 2019	Download	Release Notes
Python 3.5.7	March 18, 2019	Download	Release Notes
Python 2.7.16	March 4, 2019	Download	Release Notes
Python 3.7.2	Dec. 24, 2018	Download	Release Notes

Step 4: Scroll down the page until you find the Files option.

Step 5: Here you see a different version of python along with the operating system.

Files

Version	Operating System	Description	MD5 Sum	File Size	GPG
Gzipped source tarball	Source release		68111671e5b2db4ae77b9ab01b0f09be	23017663	SIG
XZ compressed source tarball	Source release		d33e4aa66097051c2eca45ee3604803	17133432	SIG
macOS 64-bit/32-bit installer	Mac OS X	for Mac OS X 10.6 and later	6428b4fa7583daf1a4c2c8a1cee08e6	34898416	SIG
macOS 64-bit installer	Mac OS X	for OS X 10.9 and later	5dd605c38217a45773bf5e4a936b241f	28082845	SIG
Windows help file	Windows		063999573a2c36b2ac56cade6b47c02	8131761	SIG
Windows x86-64 embeddable zip file	Windows	for AMD64/EM64T/x64	9800c3cf82fec0b1abe82184a40728a2	7504391	SIG
Windows x86-64 executable installer	Windows	for AMD64/EM64T/x64	a702b4b0af76d4b03043a583e563400	26883968	SIG
Windows x86-64 web-based installer	Windows	for AMD64/EM64T/x64	28c31c60886d73ae8e53a3bd351b4bd2	1362904	SIG
Windows x86 embeddable zip file	Windows		9fab3bd19841879fda94135174139d8	6741626	SIG
Windows x86 executable installer	Windows		33c002942a54446a3d8451a76394789	25663848	SIG
Windows x86 web-based installer	Windows		1b670cfa5d317d82c30983ea371d87c	1324608	SIG

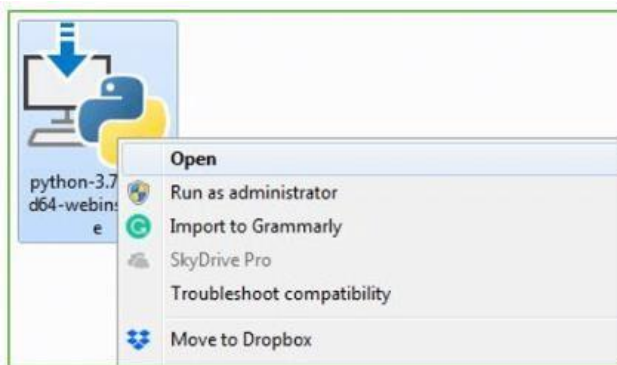
- To download Windows 32-bit python, you can select any one from the three options: Windows x86 embeddable zip file, Windows x86 executable installer or Windows x86 web-based installer.
- To download Windows 64-bit python, you can select any one from the three options: Windows x86-64 embeddable zip file, Windows x86-64 executable installer or Windows x86-64 web-based installer.

Here we will install Windows x86-64 web-based installer. Here your first part regarding which version of python is to be downloaded is completed. Now we move ahead with the second part in installing python i.e., Installation

Note: To know the changes or updates that are made in the version you can click on the Release Note Option.

Installation of Python

Step 1: Go to Download and Open the downloaded python version to carry out the installation process.



Step 2: Before you click on Install Now, make sure to put a tick on Add Python 3.7 to PATH.



Step 3: Click on Install NOW After the installation is successful. Click on Close.



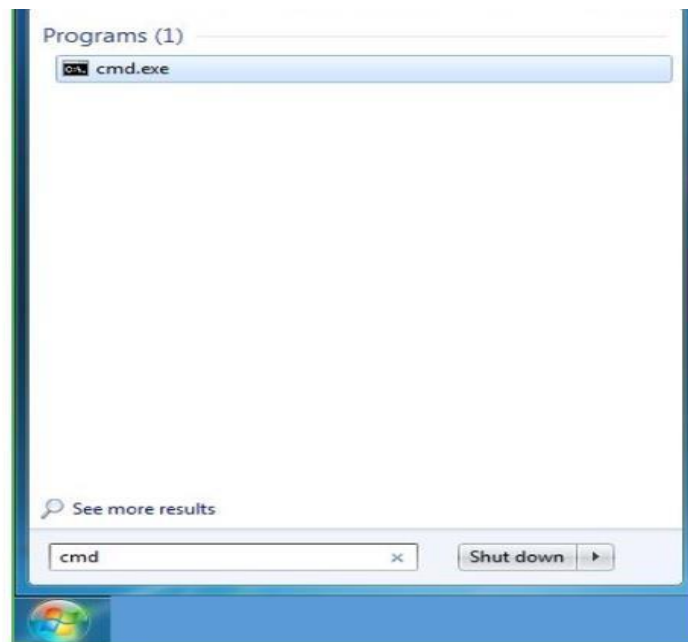
With these above three steps on python installation, you have successfully and correctly installed Python. Now is the time to verify the installation.

Note: The installation process might take a couple of minutes.

Verify the Python Installation

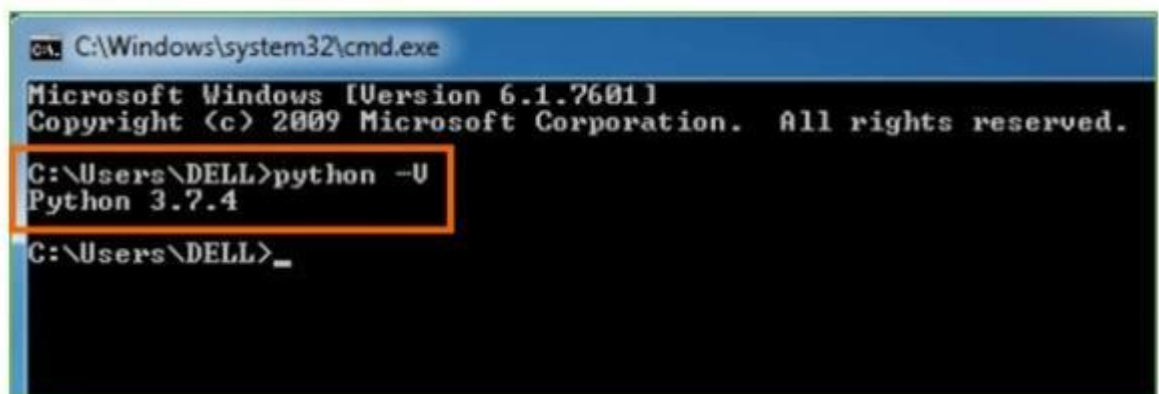
Step 1: Click on Start

Step 2: In the Windows Run Command, type “cmd”.



Step 3: Open the Command prompt option.

Step 4: Let us test whether the python is correctly installed. Type `python -V` and press Enter.



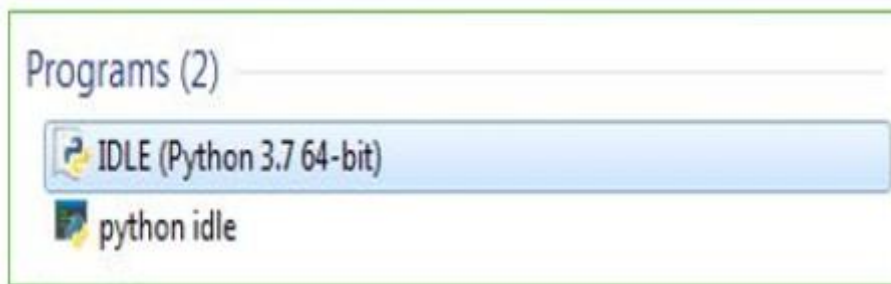
Step 5: You will get the answer as 3.7.4

Note: If you have any of the earlier versions of Python already installed. You must first uninstall the earlier version and then install the new one.

Check how the Python IDLE works

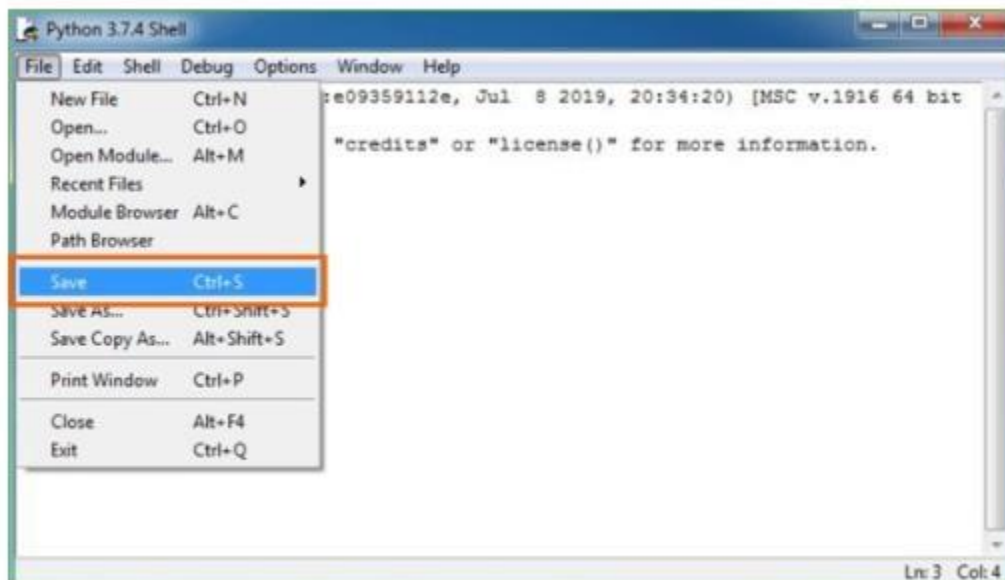
Step 1: Click on Start

Step 2: In the Windows Run command, type “python idle”.



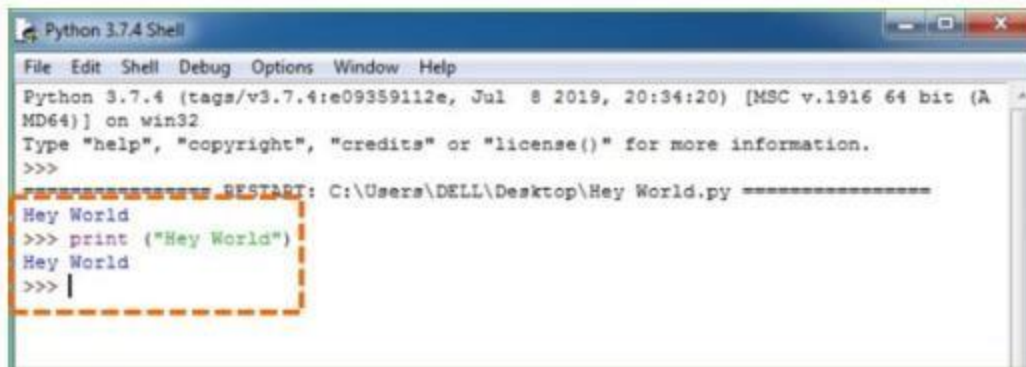
Step 3: Click on IDLE (Python 3.7 64-bit) and launch the program

Step 4: To go ahead with working in IDLE you must first save the file. Click on File > Click on Save



Step 5: Name the file and save as type should be Python files. Click on SAVE. Here I have named the files as Hey World.

Step 6: Now for e.g., enter print ("Hey World") and Press Enter.

A screenshot of a Windows command prompt window titled "Python 3.7.4 Shell". The window has a menu bar with "File", "Edit", "Shell", "Debug", "Options", "Window", and "Help". The text inside shows the Python version and build information: "Python 3.7.4 (tags/v3.7.4:e09359112e, Jul 8 2019, 20:34:20) [MSC v.1916 64 bit (AMD64)] on win32". It also displays the prompt "Type 'help', 'copyright', 'credits' or 'license()' for more information." followed by ">>>". Below this, a command is entered: "python C:\Users\DELL\Desktop\Hey World.py", which is highlighted with a dashed orange box. The output of the script is displayed: "Hey World", followed by another ">>>" prompt.

```
Python 3.7.4 Shell
File Edit Shell Debug Options Window Help
Python 3.7.4 (tags/v3.7.4:e09359112e, Jul 8 2019, 20:34:20) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
python C:\Users\DELL\Desktop\Hey World.py
Hey World
>>> print ("Hey World")
Hey World
>>> |
```

You will see that the command given is launched. With this, we end our tutorial on how to install Python. You have learned how to download python for windows into your respective operating system.

Note: Unlike Java, Python does not need semicolons at the end of the statements otherwise it won't work.

7. SYSTEM REQUIREMENTS SPECIFICATIONS

7.1 Software Requirements

The RSA-based steganography system has been designed with cross-platform compatibility as a primary consideration, ensuring accessibility across various operating systems while maintaining consistent functionality and security properties. The following software requirements define the necessary environment for proper system operation.

Operating System Compatibility: The system supports Windows 10 and newer versions, macOS 10.14 (Mojave) and above, and major Linux distributions including Ubuntu 18.04+, Fedora 30+, and Debian 10+. This broad compatibility ensures that the security solution is accessible to users regardless of their preferred operating system, maintaining consistent security properties across platforms.

Python Runtime: Python 3.7 or newer is required, as the implementation leverages modern language features including improved type hinting, enhanced asynchronous capabilities, and security improvements in the standard library. Users must have the official CPython implementation, as alternative implementations like PyPy may not fully support all the cryptographic libraries utilized by the system.

Required Libraries: The application depends on several key external libraries: the cryptography library (version 3.4.0 or newer) provides secure implementations of cryptographic algorithms including RSA encryption with appropriate padding schemes; Pillow (version 8.0.0 or newer) enables image processing operations required for steganographic embedding and extraction; NumPy (version 1.19.0 or newer) supports efficient numerical operations for image analysis and processing; and tkinter must be available for the graphical user interface components, though this is typically included with standard Python distributions.

Storage Requirements: The system requires approximately 50MB of disk space for installation, including all dependencies. Additional storage is needed for processing images, with requirements scaling based on the dimensions and number of images being processed. A minimum of 500MB of free disk space is recommended for comfortable operation with typical image sizes.

Display Requirements: A minimum display resolution of 1280x720 pixels is recommended for optimal usability of the graphical interface, though the system will function on lower resolutions with some interface elements requiring scrolling. The command-line interface has no specific display requirements.

Network Connectivity: While not strictly required for core functionality, internet connectivity may be needed for the initial installation of dependencies. Once installed, the system operates entirely offline, which is advantageous from a security perspective as it eliminates potential network-based attack vectors during operation.

Browser Requirements (for Documentation): The HTML documentation included with the system is compatible with modern web browsers including Chrome 80+, Firefox 75+, Safari 13+, and Edge 80+. The documentation uses standard HTML5 and CSS3 features without requiring JavaScript for core functionality.

Administrative Privileges: Standard user privileges are sufficient for installation when using virtual environments or user-level package installation. Administrative or root privileges are only required if installing packages system-wide, particularly on Unix-based systems.

Additional Software: No additional software is required for basic operation. For advanced features such as integration with document management systems or email clients, additional configuration may be necessary as detailed in the integration documentation.

Recommended Environment: For optimal performance when processing larger images or handling batch operations, a system with at least 4GB of RAM and a multi-core processor is recommended. The system will function on less powerful hardware, but processing times may increase proportionally.

These software requirements have been deliberately kept minimal to ensure broad accessibility while maintaining security and performance standards. The system's modular design allows for operation in various configurations, from standalone desktop applications to integrated components within larger security ecosystems.

7.2 Hardware Requirements

The hardware requirements for the RSA-based steganography system are designed to ensure reliable performance across a range of computing environments while maintaining the security integrity of the cryptographic and steganographic operations. These requirements balance accessibility with the computational demands of image processing and cryptographic operations.

Processing Requirements: The system requires a 64-bit multi-core processor with a minimum clock speed of 1.5 GHz. While the application can function on lower-powered processors, certain operations—particularly the RSA key generation and the analysis phase of adaptive steganography—may experience noticeable delays. For optimal performance when processing high-resolution images or handling batch operations, a quad-core processor with a clock speed of 2.0 GHz or higher is recommended. The system utilizes parallel processing for image analysis and steganographic operations where possible, making effective use of multiple CPU cores when available.

Memory Requirements: A minimum of 4GB of RAM is required for standard operation with typical image sizes (up to 8 megapixels). The memory requirements scale with the dimensions of the images being processed, as the system needs to maintain both the original and modified versions in memory during operations. For users working with high-resolution images (20+ megapixels) or performing batch processing operations, 8GB or more of RAM is recommended.

8. FUNCTIONAL REQUIREMENTS

8.1 User Interface & Features:

The user interface (UI) of the image steganography system is designed to be intuitive and user-friendly. It includes the following features:

1. Dashboard: A simple homepage where users can choose between encryption and decryption.
2. File Upload: Users can upload an image file where they wish to embed a secret message.
3. Text Input: A text box for users to input the secret message.
4. Encryption Button: Initiates RSA encryption and embedding into the image.
5. Decryption Button: Extracts and decrypts hidden messages from a steganographic image.
6. Download Option: Enables users to save the modified image with the hidden message.
7. Status Notifications: Displays messages confirming successful encryption, decryption, or errors.

8.2 Encryption Process Flow

The encryption process consists of the following steps:

1. User inputs the message to be hidden.
2. RSA encryption is applied using the recipient's public key.
3. The encrypted message is embedded into the image using Least Significant Bit (LSB) steganography.
4. A new steganographic image is generated and displayed for the user to download.
5. The secret message is now securely hidden within the image, protected by RSA encryption.

8.3 Decryption Process Flow:

The decryption process follows these steps:

1. User uploads the steganographic image containing the hidden message.
2. The system extracts the hidden encrypted message from the image.
3. RSA decryption is applied using the recipient's private key.
4. The original message is displayed for the user, ensuring security and confidentiality.

8.4 Performance & Security Considerations

The system ensures:

High Security: The combination of RSA encryption and steganography prevents unauthorized access.

Robustness: The embedded data remains intact despite minor image modifications.

Efficiency: Optimized algorithms ensure fast encryption and decryption without significant performance impact.

Steganalysis Resistance: Encrypted data is not directly readable, even if detected.

9. SOURCE CODE

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Image Steganography with RSA</title>
  <script src="https://cdnjs.cloudflare.com/ajax/libs/jsencrypt/3.3.2/jsencrypt.min.js"></script>
  <style>
    * {
      margin: 0;
      padding: 0;
      box-sizing: border-box;
      font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
    }

    body {
      background: linear-gradient(135deg, #3a1c71, #d76d77, #ffaf7b);
      background-size: 400% 400%;
      animation: gradient 15s ease infinite;
      height: 100vh;
      display: flex;
      flex-direction: column;
      align-items: center;
      color: white;
      padding: 20px;
    }

    @keyframes gradient {
      0% {
        background-position: 0% 50%;
      }
      50% {
        background-position: 100% 50%;
      }
      100% {
        background-position: 0% 50%;
      }
    }

    header {
```

```

    text-align: center;
    margin-bottom: 30px;
    padding-top: 30px;
}

h1 {
    font-size: 2.5rem;
    letter-spacing: 2px;
    margin-bottom: 10px;
    text-shadow: 2px 2px 4px rgba(0, 0, 0, 0.3);
}

.container {
    display: flex;
    flex-wrap: wrap;
    justify-content: center;
    gap: 50px;
    width: 100%;
    max-width: 1200px;
}

.card {
    background: rgba(255, 255, 255, 0.1);
    -webkit-backdrop-filter: blur(10px);
    backdrop-filter: blur(10px);
    border-radius: 20px;
    padding: 30px;
    width: 450px;
    display: flex;
    flex-direction: column;
    align-items: center;
    box-shadow: 0 8px 32px rgba(0, 0, 0, 0.1);
    transition: transform 0.3s, box-shadow 0.3s;
    position: relative;
    overflow: hidden;
}

.card::before {
    content: "";
    position: absolute;
    top: 0;
    left: 0;
    width: 100%;
    height: 100%;

```

```

    background: linear-gradient(45deg, rgba(255, 255, 255, 0.1), transparent);
    pointer-events: none;
}

.card:hover {
    transform: translateY(-5px);
    box-shadow: 0 15px 35px rgba(0, 0, 0, 0.2);
}

.card h2 {
    font-size: 1.8rem;
    margin-bottom: 20px;
    font-weight: 600;
}

.image-container {
    width: 200px;
    height: 200px;
    background-color: #e91e63;
    border-radius: 10px;
    margin-bottom: 20px;
    overflow: hidden;
    position: relative;
    box-shadow: 0 5px 15px rgba(0, 0, 0, 0.2);
    transition: transform 0.3s;
    display: flex;
    justify-content: center;
    align-items: center;
}

.image-container:hover {
    transform: scale(1.03);
}

.image-container img {
    width: 100%;
    height: 100%;
    object-fit: cover;
}

.upload-btn {
    background: rgba(255, 255, 255, 0.2);
    border: none;
    border-radius: 30px;

```

```

    color: white;
    padding: 12px 25px;
    font-size: 1rem;
    cursor: pointer;
    transition: all 0.3s;
    margin-bottom: 15px;
    -webkit-backdrop-filter: blur(5px);
    backdrop-filter: blur(5px);
    width: 80%;
    text-align: center;
}

.upload-btn:hover {
    background: rgba(255, 255, 255, 0.3);
    transform: translateY(-2px);
}

.action-btn {
    background: #e91e63;
    border: none;
    border-radius: 30px;
    color: white;
    padding: 12px 25px;
    font-size: 1rem;
    font-weight: 600;
    cursor: pointer;
    transition: all 0.3s;
    width: 80%;
    text-align: center;
    box-shadow: 0 5px 15px rgba(233, 30, 99, 0.4);
}

.action-btn:hover {
    background: #d81b60;
    transform: translateY(-2px);
    box-shadow: 0 8px 20px rgba(233, 30, 99, 0.6);
}

.file-input {
    display: none;
}

.message-input {
    width: 80%;

```

```

margin: 15px 0;
padding: 12px;
border-radius: 10px;
border: 1px solid rgba(255, 255, 255, 0.2);
background: rgba(255, 255, 255, 0.1);
color: white;
font-size: 0.9rem;
outline: none;
transition: all 0.3s;
}

.message-input::placeholder {
  color: rgba(255, 255, 255, 0.7);
}

.message-input:focus {
  border-color: rgba(255, 255, 255, 0.5);
  background: rgba(255, 255, 255, 0.15);
}

.message-output {
  width: 80%;
  margin: 15px 0;
  padding: 12px;
  border-radius: 10px;
  border: 1px solid rgba(255, 255, 255, 0.2);
  background: rgba(255, 255, 255, 0.1);
  color: white;
  font-size: 0.9rem;
  min-height: 60px;
  max-height: 200px;
  overflow-y: auto;
  word-break: break-word;
}

.key-input {
  width: 80%;
  margin: 15px 0;
  padding: 12px;
  border-radius: 10px;
  border: 1px solid rgba(255, 255, 255, 0.2);
  background: rgba(255, 255, 255, 0.1);
  color: white;
  font-size: 0.9rem;

```

```

    outline: none;
    transition: all 0.3s;
    height: 100px;
    resize: none;
}

#key-container {
    width: 100%;
    max-width: 800px;
    margin-top: 40px;
    padding: 20px;
    background: rgba(255, 255, 255, 0.1);
    -webkit-backdrop-filter: blur(10px);
    backdrop-filter: blur(10px);
    border-radius: 15px;
    text-align: center;
}

.key-btns {
    display: flex;
    justify-content: center;
    gap: 20px;
    margin-top: 15px;
}

.key-btn {
    background: rgba(255, 255, 255, 0.2);
    border: none;
    border-radius: 30px;
    color: white;
    padding: 10px 20px;
    font-size: 0.9rem;
    cursor: pointer;
    transition: all 0.3s;
}

.key-btn:hover {
    background: rgba(255, 255, 255, 0.3);
}

.hidden {
    display: none;
}

```



```

.key-display {
  width: 100%;
  height: 100px;
  margin-top: 15px;
  padding: 10px;
  background: rgba(0, 0, 0, 0.2);
  border-radius: 10px;
  color: #a0e4ff;
  font-family: monospace;
  font-size: 0.8rem;
  overflow-y: auto;
  text-align: left;
  white-space: pre-wrap;
  word-break: break-all;
}

.status {
  margin-top: 15px;
  font-size: 0.9rem;
  padding: 10px 15px;
  border-radius: 5px;
  background: rgba(0, 0, 0, 0.2);
  display: none;
}

.success {
  background: rgba(76, 175, 80, 0.3);
}

.error {
  background: rgba(244, 67, 54, 0.3);
}

.loading {
  display: inline-block;
  width: 20px;
  height: 20px;
  border: 3px solid rgba(255, 255, 255, 0.3);
  border-radius: 50%;
  border-top-color: white;
  animation: spin 1s ease-in-out infinite;
  margin-right: 10px;
  vertical-align: middle;
}

```

```
@keyframes spin {
  to { transform: rotate(360deg); }
}
```

```
.team-signature {
  position: absolute;
  bottom: 40px;
  font-size: 1.5rem;
  letter-spacing: 1px;
  font-weight: 300;
  opacity: 0.8;
}
```

```
@media (max-width: 950px) {
  .container {
    flex-direction: column;
    align-items: center;
  }
  .card {
    width: 90%;
    max-width: 400px;
  }
  h1 {
    font-size: 2rem;
  }
  .team-signature {
    position: static;
    margin-top: 40px;
  }
  #key-container {
    margin-bottom: 60px;
  }
}
```

```
/* Tooltip styles */
.tooltip {
  position: relative;
  display: inline-block;
}
```

```
.tooltip .tooltiptext {
  visibility: hidden;
  width: 200px;
```

```

        background-color: rgba(0, 0, 0, 0.8);
        color: #fff;
        text-align: center;
        border-radius: 6px;
        padding: 10px;
        position: absolute;
        z-index: 1;
        bottom: 125%;
        left: 50%;
        transform: translateX(-50%);
        opacity: 0;
        transition: opacity 0.3s;
        font-size: 0.8rem;
    }

    .tooltip:hover .tooltiptext {
        visibility: visible;
        opacity: 1;
    }
}
</style>
</head>
<body>
    <header>
        <h1>IMAGE STEGANOGRAPHY</h1>
    </header>

    <div class="container">
        <div class="card">
            <h2>upload</h2>
            <div class="image-container" id="encrypt-image-container">
                <img id="encrypt-image" alt="Cover image">
            </div>
            <label for="encrypt-file" class="upload-btn">Choose Image</label>
            <input type="file" id="encrypt-file" class="file-input" accept="image/*">
            <input type="text" id="encrypt-message" class="message-input" placeholder="Enter your
secret message...">
            <button id="encrypt-btn" class="action-btn">encrypt</button>
            <button id="download-encrypted-img-btn" class="upload-btn hidden">Download
Encrypted Image</button>
            <button id="download-key-btn" class="upload-btn hidden">Download Private
Key</button>
            <div id="encrypt-status" class="status"></div>
        </div>
    </div>

```

```

<div class="card">
  <h2>upload</h2>
  <div class="image-container" id="decrypt-image-container">
    <img id="decrypt-image" alt="Stego image">
  </div>
  <label for="decrypt-file" class="upload-btn">Choose Image</label>
  <input type="file" id="decrypt-file" class="file-input" accept="image/*">
  <label for="key-file" class="upload-btn">Upload Private Key File</label>
  <input type="file" id="key-file" class="file-input" accept=".txt">
  <textarea id="decrypt-key-input" class="key-input" placeholder="Or paste your private key
here..."></textarea>
  <button id="decrypt-btn" class="action-btn">decrypt</button>
  <div id="decrypt-message" class="message-output"></div>
  <div id="decrypt-status" class="status"></div>
</div>
</div>

<div id="key-container">
  <h2>RSA Key Management</h2>
  <div class="key-btns">
    <button id="generate-keys-btn" class="key-btn tooltip">
      Generate New Keys
      <span class="tooltiptext">Create new RSA key pair for encryption/decryption</span>
    </button>
    <button id="show-private-key-btn" class="key-btn tooltip">
      Show Private Key
      <span class="tooltiptext">Display your private key (keep this secret!)</span>
    </button>
    <button id="show-public-key-btn" class="key-btn tooltip">
      Show Public Key
      <span class="tooltiptext">Display your public key (share this with others)</span>
    </button>
  </div>
  <div id="key-display" class="key-display hidden"></div>
</div>

<script>
  // Initialize RSA encryption
  let crypt = new JSEncrypt({default_key_size: 1024});
  let privKey = localStorage.getItem('rsa_private_key');
  let pubKey = localStorage.getItem('rsa_public_key');

  // Generate keys if not exist
  if (!privKey || !pubKey) {

```

```

    generateNewKeys();
  } else {
    crypt.setPrivateKey(privKey);
    crypt.setPublicKey(pubKey);
  }

  // Store the current key for each encrypted image
  let currentImageKey = null;

  // DOM Elements
  const encryptImage = document.getElementById('encrypt-image');
  const encryptFile = document.getElementById('encrypt-file');
  const encryptImageContainer = document.getElementById('encrypt-image-container');
  const encryptMessage = document.getElementById('encrypt-message');
  const encryptBtn = document.getElementById('encrypt-btn');
  const encryptStatus = document.getElementById('encrypt-status');
  const downloadEncryptedImgBtn = document.getElementById('download-encrypted-img-
btn');
  const downloadKeyBtn = document.getElementById('download-key-btn');

  const decryptImage = document.getElementById('decrypt-image');
  const decryptFile = document.getElementById('decrypt-file');
  const decryptImageContainer = document.getElementById('decrypt-image-container');
  const decryptBtn = document.getElementById('decrypt-btn');
  const decryptMessage = document.getElementById('decrypt-message');
  const decryptStatus = document.getElementById('decrypt-status');
  const keyFile = document.getElementById('key-file');
  const decryptKeyInput = document.getElementById('decrypt-key-input');

  const generateKeysBtn = document.getElementById('generate-keys-btn');
  const showPrivateKeyBtn = document.getElementById('show-private-key-btn');
  const showPublicKeyBtn = document.getElementById('show-public-key-btn');
  const keyDisplay = document.getElementById('key-display');

  // Event listeners
  encryptFile.addEventListener('change', loadEncryptImage);
  decryptFile.addEventListener('change', loadDecryptImage);
  encryptBtn.addEventListener('click', encryptMessageToImage);
  decryptBtn.addEventListener('click', decryptMessageFromImage);
  generateKeysBtn.addEventListener('click', generateNewKeys);
  showPrivateKeyBtn.addEventListener('click', showPrivateKey);
  showPublicKeyBtn.addEventListener('click', showPublicKey);
  downloadEncryptedImgBtn.addEventListener('click', downloadEncryptedImage);
  downloadKeyBtn.addEventListener('click', downloadPrivateKey);

```

```

keyFile.addEventListener('change', loadKeyFile);

const decryptImage = document.getElementById('decrypt-image');
const decryptFile = document.getElementById('decrypt-file');
const decryptImageContainer = document.getElementById('decrypt-image-container');
const decryptBtn = document.getElementById('decrypt-btn');
const decryptMessage = document.getElementById('decrypt-message');
const decryptStatus = document.getElementById('decrypt-status');
const keyFile = document.getElementById('key-file');
const decryptKeyInput = document.getElementById('decrypt-key-input');

const generateKeysBtn = document.getElementById('generate-keys-btn');
const showPrivateKeyBtn = document.getElementById('show-private-key-btn');
const showPublicKeyBtn = document.getElementById('show-public-key-btn');
const keyDisplay = document.getElementById('key-display');

// Event listeners
encryptFile.addEventListener('change', loadEncryptImage);
decryptFile.addEventListener('change', loadDecryptImage);
encryptBtn.addEventListener('click', encryptMessageToImage);
decryptBtn.addEventListener('click', decryptMessageFromImage);
generateKeysBtn.addEventListener('click', generateNewKeys);
showPrivateKeyBtn.addEventListener('click', showPrivateKey);
showPublicKeyBtn.addEventListener('click', showPublicKey);
downloadEncryptedImgBtn.addEventListener('click', downloadEncryptedImage);
downloadKeyBtn.addEventListener('click', downloadPrivateKey);
keyFile.addEventListener('change', loadKeyFile);

reader.readAsDataURL(file);
encryptStatus.style.display = 'none';
downloadEncryptedImgBtn.classList.add('hidden');
downloadKeyBtn.classList.add('hidden');

// Clear the message input for a fresh start
encryptMessage.value = "";
}

// Generate a new key for each image
function generateNewKeyForImage() {
  const imageCrypt = new JSEncrypt({default_key_size: 1024});
  imageCrypt.getKey();

  currentImageKey = imageCrypt.getPrivateKey();

```

```

const imagePublicKey = imageCrypt.getPublicKey();

// Use this key for the current image encryption
crypt.setPrivateKey(currentImageKey);
crypt.setPublicKey(imagePublicKey);

showStatus(encryptStatus, 'New encryption key generated for this image', 'success');
setTimeout(() => { encryptStatus.style.display = 'none'; }, 3000);
}

// Load image for decryption
function loadDecryptImage(e) {
  const file = e.target.files[0];
  if (!file) return;

  const reader = new FileReader();
  reader.onload = function(event) {
    decryptImage.src = event.target.result;
    decryptImageContainer.style.backgroundColor = 'transparent';
  };
  reader.readAsDataURL(file);
  decryptStatus.style.display = 'none';
  decryptMessage.innerText = "";

  // Clear the key input for new decryption
  decryptKeyInput.value = "";
}

// Load key file
function loadKeyFile(e) {
  const file = e.target.files[0];
  if (!file) return;

  const reader = new FileReader();
  reader.onload = function(event) {
    decryptKeyInput.value = event.target.result;
  };
  reader.readAsText(file);
}

// Encrypt message into image
function encryptMessageToImage() {
  if (!encryptImage.src) {
    showStatus(encryptStatus, 'Please select an image first', 'error');
  }
}

```

```

    return;
}

const message = encryptMessage.value.trim();
if (!message) {
    showStatus(encryptStatus, 'Please enter a message to encrypt', 'error');
    return;
}

showStatus(encryptStatus, '<div class="loading"></div> Encrypting message...', "");

// Create a canvas element
const canvas = document.createElement('canvas');
const ctx = canvas.getContext('2d');

// Create a new image to load the source
const img = new Image();
img.crossOrigin = 'Anonymous';

img.onload = function() {
    // Set canvas dimensions to match the image
    canvas.width = img.width;
    canvas.height = img.height;

    // Draw the image on the canvas
    ctx.drawImage(img, 0, 0);

    // Encrypt the message with RSA
    const encryptedMessage = crypt.encrypt(message);
    if (!encryptedMessage) {
        showStatus(encryptStatus, 'Encryption failed. Try again or generate new keys.', 'error');
        return;
    }

    // Convert the encrypted message to binary
    const binaryMessage = convertToBinary(encryptedMessage);

    // Get image data
    const imageData = ctx.getImageData(0, 0, canvas.width, canvas.height);
    const data = imageData.data;

    // Check if the image is large enough for the message
    if (data.length / 4 < binaryMessage.length) {

```



```

        showStatus(encryptStatus, 'Image too small for this message. Choose a larger image.',
'error');
        return;
    }

    // Embed the message length at the beginning
    const messageLengthBinary = convertToBinary(binaryMessage.length.toString());
    for (let i = 0; i < 32; i++) {
        const bit = i < messageLengthBinary.length ? messageLengthBinary[i] : '0';
        // Modify the least significant bit of the red channel
        data[i * 4] = (data[i * 4] & 254) | parseInt(bit);
    }

    // Embed the message
    for (let i = 0; i < binaryMessage.length; i++) {
        // Start after the length bits (32 pixels)
        const position = (i + 32) * 4;
        if (position >= data.length) break;

        // Modify the least significant bit of the blue channel
        data[position + 2] = (data[position + 2] & 254) | parseInt(binaryMessage[i]);
    }

    // Put the modified image data back to the canvas
    ctx.putImageData(imageData, 0, 0);

    // Store the encrypted image
    encryptedImageData = canvas.toDataURL('image/png');

    // Display success and show download buttons
    showStatus(encryptStatus, 'Message encrypted successfully!', 'success');
    downloadEncryptedImgBtn.classList.remove('hidden');
    downloadKeyBtn.classList.remove('hidden');
};

img.onerror = function() {
    showStatus(encryptStatus, 'Error loading image. Try another image.', 'error');
};

img.src = encryptImage.src;
}

// Function to download the encrypted image
function downloadEncryptedImage() {

```

```

if (!encryptedImageData) {
    showStatus(encryptStatus, 'No encrypted image available.', 'error');
    return;
}

const link = document.createElement('a');
link.href = encryptedImageData;
link.download = 'encrypted.png';
document.body.appendChild(link);
link.click();
document.body.removeChild(link);
}

// Function to download the private key
function downloadPrivateKey() {
    if (!currentImageKey) {
        showStatus(encryptStatus, 'No private key available for this image.', 'error');
        return;
    }

    const blob = new Blob([currentImageKey], { type: 'text/plain' });
    const link = document.createElement('a');
    link.href = URL.createObjectURL(blob);
    link.download = 'privatekey.txt';
    document.body.appendChild(link);
    link.click();
    document.body.removeChild(link);
    URL.revokeObjectURL(link.href);
}

// Decrypt message from image
function decryptMessageFromImage() {
    if (!decryptImage.src) {
        showStatus(decryptStatus, 'Please select an image first', 'error');
        return;
    }

    // Get the private key from input or file
    const key = decryptKeyInput.value.trim();
    if (!key) {
        showStatus(decryptStatus, 'Please enter or upload a private key', 'error');
        return;
    }
}

```

```

showStatus(decryptStatus, '<div class="loading"></div> Decrypting message...', "");

// Set the private key for decryption
const decrypter = new JSEncrypt();
decrypter.setPrivateKey(key);

// Create a canvas element
const canvas = document.createElement('canvas');
const ctx = canvas.getContext('2d');

// Create a new image to load the source
const img = new Image();
img.crossOrigin = 'Anonymous';

img.onload = function() {
  // Set canvas dimensions to match the image
  canvas.width = img.width;
  canvas.height = img.height;

  // Draw the image on the canvas
  ctx.drawImage(img, 0, 0);

  // Get image data
  const imageData = ctx.getImageData(0, 0, canvas.width, canvas.height);
  const data = imageData.data;

  // Extract the message length first (from first 32 pixels)
  let messageLengthBinary = "";
  for (let i = 0; i < 32; i++) {
    messageLengthBinary += (data[i * 4] & 1).toString();
  }

  const messageLength = parseInt(convertFromBinary(messageLengthBinary));

  if (isNaN(messageLength) || messageLength <= 0 || messageLength > data.length / 4) {
    showStatus(decryptStatus, 'No hidden message found in this image.', 'error');
    return;
  }

  // Extract the binary message
  let binaryMessage = "";
  for (let i = 0; i < messageLength; i++) {
    // Start after the length bits (32 pixels)
    const position = (i + 32) * 4;

```

```

        if (position >= data.length) break;

        // Extract the least significant bit of the blue channel
        binaryMessage += (data[position + 2] & 1).toString();
    }

    // Convert binary to text
    const encryptedMessage = convertFromBinary(binaryMessage);

    // Decrypt the message with RSA
    const decryptedMessage = decrypter.decrypt(encryptedMessage);

    if (!decryptedMessage) {
        showStatus(decryptStatus, 'Decryption failed. Make sure you have the correct private
key.', 'error');
        decryptMessage.innerText = 'Decryption failed. Please check your key.';
        return;
    }

    // Display the decrypted message
    decryptMessage.innerText = decryptedMessage;
    showStatus(decryptStatus, 'Message decrypted successfully!', 'success');
};

img.onerror = function() {
    showStatus(decryptStatus, 'Error loading image. Try another image.', 'error');
};

img.src = decryptImage.src;
}

// Helper function to show status messages
function showStatus(element, message, type) {
    element.innerHTML = message;
    element.className = 'status';
    if (type) element.classList.add(type);
    element.style.display = 'block';
}

// Helper function to convert text to binary
function convertToBinary(text) {
    let binary = "";
    for (let i = 0; i < text.length; i++) {
        const charCode = text.charCodeAt(i);

```

```

        const binaryChar = charCode.toString(2).padStart(8, '0');
        binary += binaryChar;
    }
    return binary;
}

// Helper function to convert binary to text
function convertFromBinary(binary) {
    let text = "";
    for (let i = 0; i < binary.length; i += 8) {
        const byte = binary.substr(i, 8);
        if (byte.length < 8) break;
        const charCode = parseInt(byte, 2);
        text += String.fromCharCode(charCode);
    }
    return text;
}

// Generate new RSA key pair
function generateNewKeys() {
    crypt = new JSEncrypt({default_key_size: 1024});
    crypt.getKey();

    privKey = crypt.getPrivateKey();
    pubKey = crypt.getPublicKey();

    // Also update current image key
    currentImageKey = privKey;

    // Save keys to localStorage
    localStorage.setItem('rsa_private_key', privKey);
    localStorage.setItem('rsa_public_key', pubKey);

    keyDisplay.innerText = 'New keys generated and saved!';
    keyDisplay.classList.remove('hidden');
    setTimeout(() => {
        keyDisplay.classList.add('hidden');
    }, 3000);
}

// Show private key
function showPrivateKey() {
    if (currentImageKey) {
        keyDisplay.innerText = currentImageKey; // Show the current image's key
    }
}

```

```

    } else if (privKey) {
        keyDisplay.innerText = privKey;
    } else {
        keyDisplay.innerText = 'No private key found. Generate keys first.';
    }
    keyDisplay.classList.remove('hidden');
}

// Show public key
function showPublicKey() {
    if (!pubKey) {
        keyDisplay.innerText = 'No public key found. Generate keys first.';
    } else {
        keyDisplay.innerText = pubKey;
    }
    keyDisplay.classList.remove('hidden');
}
</script>
</body>
</html>

```

10.RESULTS AND DISCUSSION

9.1 Implementation Description:

The implementation of the image steganography system is done using Python, leveraging libraries such as OpenCV, NumPy, and Cryptography. The core functionalities include:

RSA Key Generation: Generates a public-private key pair for secure encryption.

Image Processing: Loads, modifies, and saves images using OpenCV.

Message Embedding: Uses LSB substitution to hide encrypted text in the image.

Message Extraction: Retrieves and decrypts hidden messages securely.

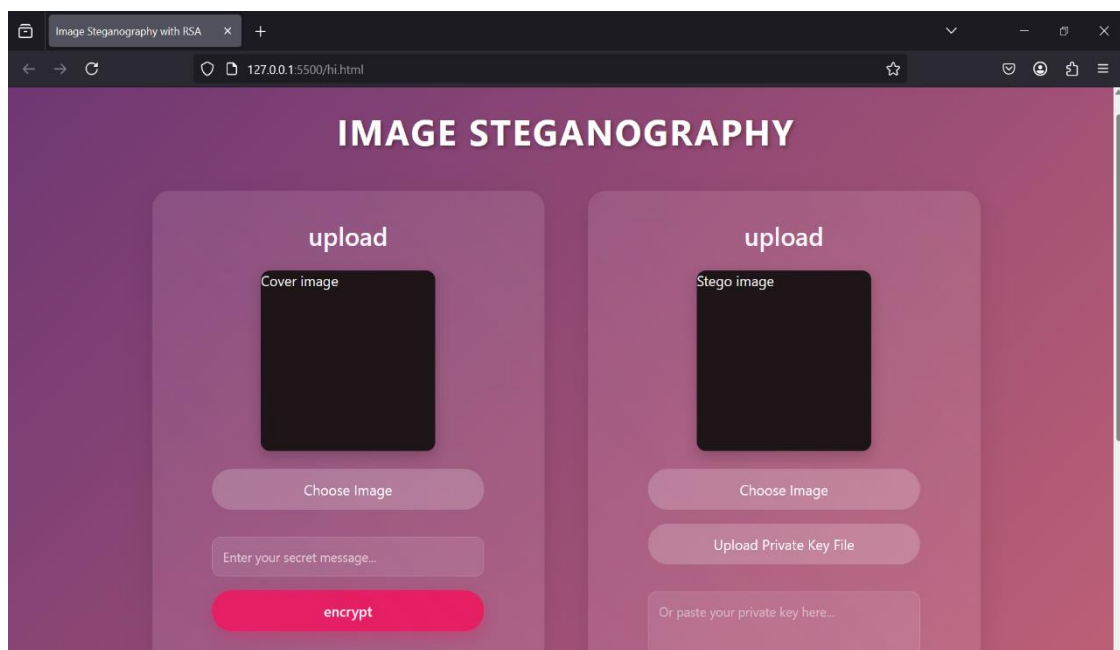
User Interaction: Simple UI for encryption and decryption operations.

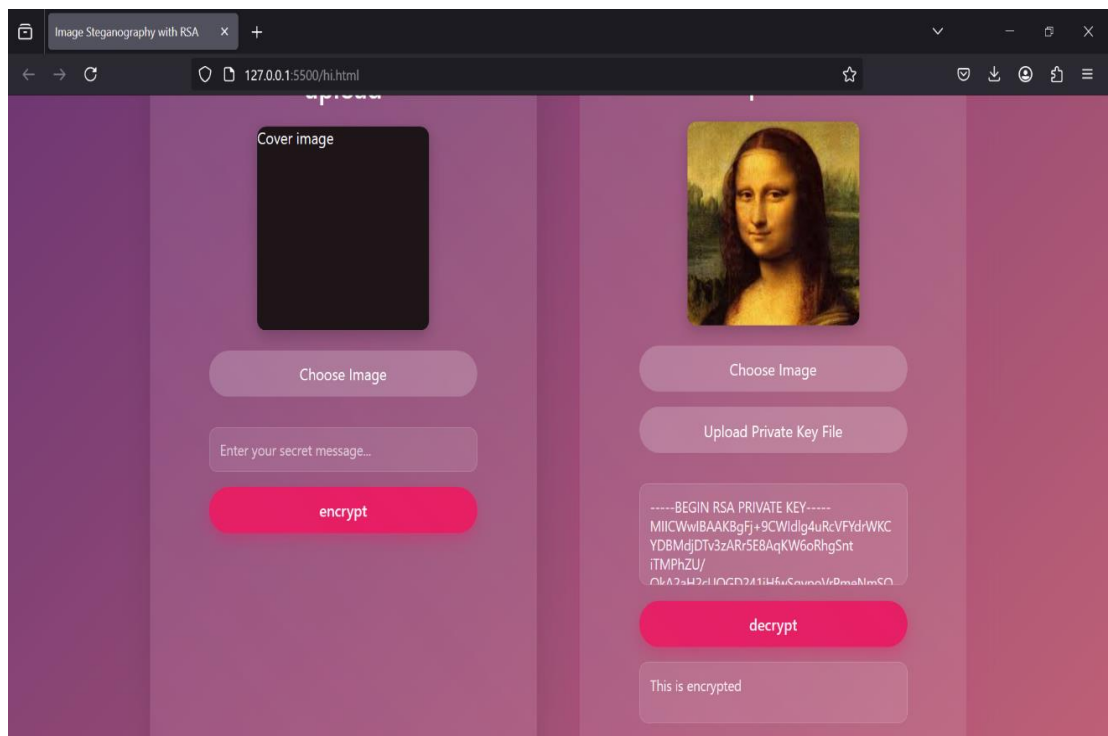
9.2 Image Encryption & Decryption:

Encryption Process: The user inputs text, which is encrypted with RSA and embedded into an image.

Decryption Process: The hidden message is extracted and decrypted back into readable text.

Testing Results: The system successfully encrypts and decrypts messages without visible image distortion, ensuring security and reliability.





11. CONCLUSION AND FUTURE SCOPE

Conclusion:

The combination of steganography and RSA encryption provides a highly secure method for hidden communication. The system successfully ensures confidentiality, integrity, and robustness, making it an effective solution for secure data transmission. Future improvements may include support for other file formats and enhanced steganalysis resistance. Image steganography using the RSA algorithm provides a robust method for secure data hiding by combining cryptography and steganography. The RSA algorithm ensures the confidentiality and integrity of hidden data by encrypting it before embedding it into an image. This dual-layer security approach enhances resistance to attacks, making it difficult for unauthorized users to retrieve or manipulate the concealed information. By leveraging the strengths of RSA encryption and image steganography, this technique can be effectively applied in secure communications, watermarking, and digital forensics. However, trade-offs such as increased computational complexity and potential distortion of the cover image should be carefully considered. Future improvements can focus on optimizing embedding techniques and enhancing security against steganalysis attacks.

References:

1. *Steganography*, 2020, [online] Available: <https://en.wikipedia.org/wiki/Steganography>.
Show in Context [Google Scholar](#)
2. H. Shi, X.-Y. Zhang, S. Wang, G. Fu and J. Tang, "Synchronized detection and recovery of steganographic messages with adversarial learning", *Proc. Int. Conf. Comput. Sci.*, pp. 31-43, 2019.
Show in Context [CrossRef](#) [Google Scholar](#)
3. N. F. Hordri, S. S. Yuhani and S. M. Shamsuddin, "Deep learning and its applications: A review", *Proc. Conf. Postgraduate Annu. Res. Informat. Seminar*, pp. 1-6, 2016.
Show in Context [Google Scholar](#)
4. N. F. Johnson and S. Jajodia, "Exploring steganography: Seeing the unseen", *Computer*, vol. 31, no. 2, pp. 26-34, Feb. 1998.
Show in Context [View Article](#)
[Google Scholar](#)
5. S. Gupta, G. Gujral and N. Aggarwal, "Enhanced least significant bit algorithm for image steganography", *Int. J. Comput. Eng. Manage.*, vol. 15, no. 4, pp. 40-42, 2012.
Show in Context [Google Scholar](#)
6. R. Das and T. Tuithung, "A novel steganography method for image based on Huffman encoding", *Proc. 3rd Nat. Conf. Emerg. Trends Appl. Comput. Sci.*, pp. 14-18, Mar. 2012.
Show in Context [View Article](#)
[Google Scholar](#)
7. A. Singh and H. Singh, "An improved LSB based image steganography technique for RGB images", *Proc. IEEE Int. Conf. Electr. Comput. Commun. Technol. (ICECCT)*, pp. 1-4, Mar. 2015.
Show in Context [View Article](#)

[Google Scholar](#)

8.Z. Qu, Z. Cheng, W. Liu and X. Wang, "A novel quantum image steganography algorithm based on exploiting modification direction", *Multimedia Tools Appl.*, vol. 78, no. 7, pp. 7981-8001, Apr. 2019.

Show in Context [CrossRef](#) [Google Scholar](#)

9.S. Wang, J. Sang, X. Song and X. Niu, "Least significant qubit (LSQb) information hiding algorithm for quantum image", *Measurement*, vol. 73, pp. 352-359, Sep. 2015.

Show in Context [CrossRef](#) [Google Scholar](#)

10.N. Patel and S. Meena, "LSB based image steganography using dynamic key cryptography", *Proc. Int. Conf. Emerg. Trends Commun. Technol. (ETCT)*, pp. 1-5, Nov. 2016.

Show in Context [View Article](#)

[Google Scholar](#)

11.O. Elharrouss, N. Almaadeed and S. Al-Maadeed, "An image steganography approach based on k-least significant bits (k-LSB)", *Proc. IEEE Int. Conf. Informat. IoT Enabling Technol. (ICIOT)*, pp. 131-135, Feb. 2020.

Show in Context [View Article](#)

[Google Scholar](#)

12.M. V. S. Tarun, K. V. Rao, M. N. Mahesh, N. Srikanth and M. Reddy, "Digital video steganography using LSB technique", *Red*, vol. 100111, Apr. 2020.

Show in Context [Google Scholar](#)

13.S. S. M. Than, "Secure data transmission in video format based on LSB and Huffman coding", *Int. J. Image Graph. Signal Process.*, vol. 12, no. 1, pp. 10, 2020.

Show in Context [CrossRef](#) [Google Scholar](#)

14.M. B. Tuieb, M. Z. Abdullah and N. S. Abdul-Razaq, "An efficiency secured and reversible video steganography approach based on lest significant", *J. Cellular Automata*, vol. 16, no. 17, Apr. 2020.

Show in Context [Google Scholar](#)

15.R. J. Mstafa, K. M. Elleithy and E. Abdelfattah, "A robust and secure video steganography method in DWT-DCT domains based on multiple object tracking and ECC", *IEEE Access*, vol. 5, pp. 5354-5365, 2017.

Show in Context [View Article](#)

[Google Scholar](#)

16.K. A. Al-Afandy, O. S. Faragallah, A. Elmhawwy, E.-S.-M. El-Rabaie and G. M. El-Banby, "High security data hiding using image cropping and LSB least significant bit steganography", *Proc. 4th IEEE Int. Colloq. Inf. Sci. Technol. (CiSt)*, pp. 400-404, Oct. 2016.

Show in Context [View Article](#)

[Google Scholar](#)

17.A. Arya and S. Soni, "Performance evaluation of secrete image steganography techniques using least significant bit (LSB) method", *Int. J. Comput. Sci. Trends Technol.*, vol. 6, no. 2, pp. 160-165, 2018.

Show in Context [Google Scholar](#)

18.G. Swain, "Very high capacity image steganography technique using quotient value differencing and LSB substitution", *Arabian J. Sci. Eng.*, vol. 44, no. 4, pp. 2995-3004, Apr. 2019.

Show in Context [CrossRef](#) [Google Scholar](#)

19.A. Qiu, X. Chen, X. Sun, S. Wang and W. Guo, "Coverless image steganography method based on feature selection", *J. Inf. Hiding Privacy Protection*, vol. 1, no. 2, pp. 49, 2019.

Show in Context [CrossRef](#) [Google Scholar](#)

20.R. D. Rashid and T. F. Majeed, "Edge based image steganography: Problems and solution", *Proc. Int. Conf. Commun. Signal Process. Appl. (ICCSPA)*, pp. 1-5, Mar. 2019.

Show in Context [View Article](#)

[Google Scholar](#)

21.X. Liao, J. Yin, S. Guo, X. Li and A. K. Sangaiah, "Medical JPEG image steganography based on preserving inter-block dependencies", *Comput. Electr. Eng.*, vol. 67, pp. 320-329, Apr. 2018.

Show in Context [CrossRef](#) [Google Scholar](#)

22.W. Lu, Y. Xue, Y. Yeung, H. Liu, J. Huang and Y. Shi, "Secure halftone image steganography based on pixel density transition", *IEEE Trans. Dependable Secure Comput.*, Aug. 2019.

Show in Context [View Article](#)

[Google Scholar](#)

23.Y. Zhang, C. Qin, W. Zhang, F. Liu and X. Luo, "On the fault-tolerant performance for a class of robust image steganography", *Signal Process.*, vol. 146, pp. 99-111, May 2018.

Show in Context [CrossRef](#) [Google Scholar](#)

24.H. M. Sidqi and M. S. Al-Ani, "Image steganography: Review study", *Proc. Int. Conf. Image Process. Comput. Vis. Pattern Recognit. (IPCV)*, pp. 134-140, 2019.

Show in Context [Google Scholar](#)

25.P. Wu, Y. Yang and X. Li, "Image-into-image steganography using deep convolutional network", *Proc. Pacific Rim Conf. Multimedia*, pp. 792-802, 2018.

Show in Context [CrossRef](#) [Google Scholar](#)

26.P. Wu, Y. Yang and X. Li, "StegNet: Mega image steganography capacity with deep convolutional network", *Future Internet*, vol. 10, no. 6, pp. 54, Jun. 2018.

Show in Context [CrossRef](#) [Google Scholar](#)

27.X. Duan, K. Jia, B. Li, D. Guo, E. Zhang and C. Qin, "Reversible image steganography scheme based on a U-Net structure", *IEEE Access*, vol. 7, pp. 9314-9323, 2019.

Show in Context [View Article](#)

[Google Scholar](#)

28.T. P. Van, T. H. Dinh and T. M. Thanh, "Simultaneous convolutional neural network for highly efficient image steganography", *Proc. 19th Int. Symp. Commun. Inf. Technol. (ISCIT)*, pp. 410-415, Sep. 2019.

Show in Context [View Article](#)

[Google Scholar](#)

29.R. Rahim and S. Nadeem, "End-to-end trained CNN encoder-decoder networks for image steganography", *Proc. Eur. Conf. Comput. Vis. (ECCV)*, pp. 1-6, 2018.

Show in Context [Google Scholar](#)

30.Z. Wang, N. Gao, X. Wang, J. Xiang and G. Liu, "STNet: A style transformation network for deep image steganography", *Proc. Int. Conf. Neural Inf. Process*, pp. 3-14, 2019.

Show in Context [CrossRef](#) [Google Scholar](#)