WWW.IJRPR.COM

International Journal of Research Publication and Reviews

Journal homepage: www.ijrpr.com ISSN 2582-7421

End-to-End Secure Image Steganography using RSA Encryption

Subramanian P L[1], Baruni M K[2] , Laksitha N R G[3]

[1]Assistant Professor, Department of Information Technology, K.L.N. College of Engineering, Sivaganga – 630612, Tamil Nadu, India.
[2,3],UG Scholar, Department of Information Technology, K.L.N. College of Engineering, Sivaganga – 630612, Tamil Nadu, India
pls.mecse@gmail.com, barunibaruni6@gmail.com, laksitha.n.r.g@gmail.com

ABSTRACT

- Steganography is the practice of concealing information within another medium to ensure secure communication without raising suspicion. Unlike cryptography, which focuses on encrypting data to make it unreadable, steganography aims to hide the existence of the message itself. This technique has been employed throughout history, from ancient civilizations using invisible inks to modern digital applications involving multimedia files such as images, audio, and video. In the digital era, steganography has gained significant importance due to increasing concerns over data privacy and cybersecurity. Various steganographic techniques exist, including spatial domain methods (e.g., Least Significant Bit (LSB) substitution), transform domain techniques (e.g., Discrete Cosine Transform (DCT) and Discrete Wavelet Transform (DWT)), and adaptive steganography, which intelligently hides data based on image content. These techniques differ in terms of robustness, payload capacity, and imperceptibility. This project's Sustainable Development Goal is to promote the creation of cutting-edge AI/ML technologies for data protection and digital forensics and Quality Education by promoting awareness and understanding of cybersecurity and data protection.

KeyWords: Datahiding, Decoding, Information Concealment, Detection, Embedding, Cipher text, cryptography key.

1.INTRODUCTION

Steganography is the art and science of hiding information within non-secret, ordinary data in such a way that the presence of the hidden message is concealed. Unlike cryptography, which protects the content of a message, steganography hides the fact that a message even exists. Common carriers used for steganography include digital images, audio files, video, and text. With the rapid advancement of digital communication and multimedia technologies, steganography has become a crucial tool for secure communication, but it also presents challenges in terms of misuse for illegal data transmission.

The prediction of steganography refers to the use of computational techniques—often involving machine learning and image processing—to detect or forecast the presence of hidden information within digital media. Predictive models aim to analyze features of the media, such asstatistical anomalies or patterns in pixel values, to accurately identify whether steganography.

As steganography techniques become more sophisticated, developing reliable and intelligent prediction systems is essential for maintaining information security. This research or study explores various methodologies and tools used in predicting steganographic content, evaluating their accuracy, efficiency, and robustness.

2.METHODOLOGY

This project implements a browser-based tool for image-based steganography integrated with RSA encryption. The methodology follows a step-by-step pipeline that combines cryptographic security and least significant bit (LSB) steganography to hide and retrieve secret messages inside digital images. The approach is divided into the following phases:

1. User Interface Design

The frontend interface is designed using HTML and CSS to provide an intuitive user experience. Key UI components include:

- File input fields to upload cover images and stego-images.
- Text input for the secret message.
- RSA public/private key display boxes.
- A toggle option for enabling or disabling encryption.

- Canvas elements for rendering images before and after encoding.

2. RSA Key Generation

RSA encryption is integrated to secure the message prior to embedding. Upon clicking the Generate Keys button, the tool:

- Uses the Forge.js library to create a 1024-bit RSA key pair.
- Displays the public key (used for encryption) and private key (used for decryption).

This ensures that even if the stego-image is intercepted, the message remains confidential without the private key.

3. Message Encryption

If the encryption toggle is enabled:

- The plaintext message is encrypted using the RSA public key.
- The encrypted binary is base64-encoded to ensure safe conversion to binary bits.
- This step adds an extra layer of protection before the message is hidden within the image.

4. Message Embedding (LSB Encoding)

- The actual steganography process uses the Least Significant Bit (LSB) technique:
- The selected image is loaded into an HTML canvas.
- The message (plain or encrypted) is converted to a binary string.
- The first 32 pixels store the length of the binary message.
- The message bits are then inserted into the least significant bit of each byte in the image's pixel data.
- The modified image is then displayed on the canvas, and can be downloaded as a PNG file.

5. Message Extraction (LSB Decoding)

To decode a hidden message from an image:

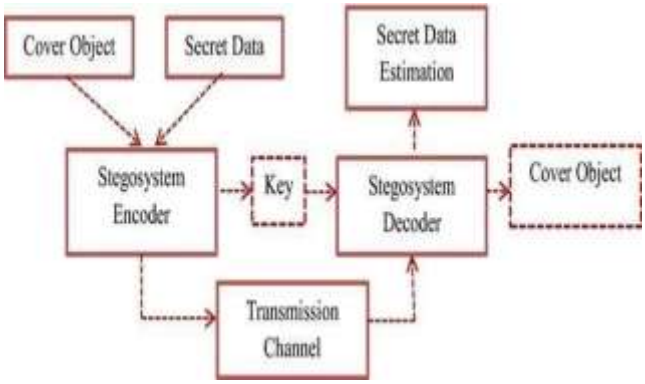
- The stego-image is uploaded and rendered on a second canvas.
- The first 32 bytes are read to determine the length of the hidden message.
- The next bits are extracted and grouped into 8-bit segments to reconstruct characters.
- If encryption was used, the RSA private key is required to decrypt the message.
- A “Show Encrypted” feature is also provided to display the raw encrypted message without decryption.

6. Data Prediction and Analysis

While this tool focuses on steganography and encryption, the foundation can support further prediction-based extensions:

- Detection of encoded vs. non-encoded images using statistical analysis.
- Prediction algorithms (e.g., ML models) to identify suspicious LSB patterns.
- This can be a next-phase goal for applying predictive modeling in steganalysis.

3.MODELING AND ANALYSIS



System Modeling Overview

The proposed system is designed to provide a secure, end-to-end steganographic solution that combines **Least Significant Bit (LSB)** image steganography with **RSA public-key encryption**. The entire architecture is client-side, implemented in JavaScript, and runs in the web browser without any backend dependency. The core modeling approach follows modular software engineering principles and uses the **Incremental Software Development Model**, which supports continuous testing and enhancement of each module.

Software Development Life Cycle (SDLC) Model

This project adopts the **Incremental Model**, which allows development in functional stages. Each increment includes complete SDLC phases and delivers a fully functional feature, such as key generation, encryption, message encoding, and decoding.

Reasons for Choosing the Incremental Model:

- Facilitates early delivery of core features.
- Supports independent testing and debugging of each module.
- Encourages iterative improvement based on feedback and user experience.
- Ideal for research-based systems with changing requirements like stego-detection and encryption methods.

Functional and Non-Functional Requirements

Functional Requirements

Functionality	Description
Key Generation	Generates 1024-bit RSA key pairs using Forge.js.
Message Encryption	Encrypts input message using RSA public key.
Steganography Encoding	Embeds (plain or encrypted) message into image using LSB method.
Image Upload	Accepts PNG/JPG image files for encoding/decoding.
Decoding and Decryption	Extracts hidden message and decrypts it using the private key.
Encrypted Preview	Shows raw encrypted data from image without decryption.
Download Function	Allows encoded image to be saved locally.

Non-Functional Requirements

- **Security:** Ensures confidentiality using RSA encryption.
- **Usability:** Intuitive, responsive UI for technical and non-technical users.
- **Performance:** Fast client-side execution with minimal latency.
- **Portability:** Works across platforms (desktop, mobile) using modern browsers.
- **Scalability:** Easily extendable to audio/video steganography in future.

- **Reliability:** Tested for accuracy, error handling, and message recovery

System Architecture

The system is composed of five key modules:

Module	Description
Key Generation Module	Uses Forge.js to create RSA key pairs and displays them to the user.
Encryption Module	Encrypts the input text message using the RSA public key.
Encoding Module	Converts message to binary and embeds it into the LSBs of the image pixels.
Decoding Module	Extracts binary data from the image and reconstructs the original message.
Encrypted Preview	Allows user to see raw encrypted message without decrypting.

Data Flow Representation

The **Data Flow Diagram (DFD)** represents the logical flow from user input to system output.

- **Input:** Image file + Text message + (Optional) RSA keys
- **Processes:** Key Generation → Encryption → LSB Embedding → Download
- **Output:** Encoded image file, Decoded message
- **Storage:** Client-side only; no server interaction

UML Design Overview

The system is modeled using the following **UML diagrams** to ensure complete design visualization:

- **Use Case Diagram:** Shows user interactions (e.g., generate keys, encode, decode, download).
- **Sequence Diagram:** Models the timeline of function calls between UI, encryption, and steganography logic.
- **Activity Diagram:** Describes workflows from input collection to encoded output.
- **Class Diagram:** Defines class responsibilities for modules like RSAHandler, ImageProcessor, UIManager.
- **Collaboration Diagram:** Maps interactions between functions using numbered messages for clarity.

Technological Stack and Implementation Layers

- **HTML5:** Interface structure (file input, text input, buttons, canvas display)
- **CSS3:** Responsive design, layout styling, and user experience
- **JavaScript (Vanilla):** Core logic for encryption, image processing, and steganography
- **Forge.js:** Library used for RSA key generation and encryption
- **Canvas API:** Handles pixel-level image manipulation for LSB encoding

4. RESULTS AND DISCUSSION

The proposed system was successfully developed and tested in a fully client-side browser environment. It integrates RSA public-key encryption with Least Significant Bit (LSB) steganography to allow secure message hiding within image files. The system met all core functionality objectives, including key generation, message encryption, image encoding, and decoding.

Key outcomes:

- RSA key pair (1024-bit) was reliably generated using Forge.js.
- Secret messages were successfully encrypted and embedded in carrier images using LSB.
- Encoded images maintained visual integrity and were indistinguishable from originals to the human eye.
- Decoding and decryption retrieved the original message accurately using the corresponding private key.

- The system functioned entirely offline within the browser, ensuring user privacy.

Observations from User Interface

1. Message Encoding Phase

   Users could upload an image, input a message, and optionally apply RSA encryption. Upon clicking "Encode", the message was embedded using LSB without altering the visual fidelity of the image.

2. Key Management

   The interface clearly displayed the generated RSA public and private keys. Users could copy or reuse them for encryption and decryption workflows.

3. Decoding Phase

   Uploaded encoded images were processed, and hidden messages were accurately extracted and, if encrypted, decrypted using the provided private key.

4. User Feedback and Error Handling
5. Validation prompts ensured:
   - Alerts when files or keys were missing.
   - Warnings for oversized messages relative to image capacity.
   - Clear failure messages during decryption errors.

Test case result:

S. No	Test Case	Input	Expected Output	Actual Result	Status
1	RSA Key Generation	Click “Generate Keys”	Valid public/private key shown	Keys displayed	Pass
2	Encrypt & Decrypt Message	Text = "hello"	Decrypted = "hello"	Correctly decrypted	Pass
3	Encode Without Encryption	Plain message + image	Message embedded, image unchanged	As expected	Pass
4	Encode With Encryption	Message + RSA Public Key + Image	Encrypted message hidden in image	Securely encoded	Pass
5	Decode With Private Key	Encoded image + Private Key	Message recovered	Accurate message	Pass
6	Decode Without Private Key	Encoded image only	Encrypted content shown	Encrypted data shown	Pass
7	Oversized Message	Long text + Small image	Error alert: message too long	Error displayed	Pass
8	No Image Uploaded	Click encode with no image selected	Alert for missing image	Alert shown	Pass

Fig 4.1 UI where user selects image, enters text , and RSA keys are shown

Fig:4.2 User uploads an encode image to decode hidden message

Fig:4.3 Decoded hidden text message and image is displayed

Fig 4.4 Encoded image is shown with download option

5.CONCLUSION

The project successfully demonstrates a secure and efficient method for hiding sensitive information within images using a combination of RSA encryption and LSB-based steganography. By integrating public-key cryptography with image-level data embedding, the system ensures both confidentiality and invisibility of the hidden message. The fully client-side implementation protects user data from external threats, while the intuitive interface allows even non-technical users to perform secure encoding and decoding. This approach highlights the practical application of steganography in modern cybersecurity, enabling private communication without raising suspicion.

6.REFERENCES

1. Menezes, A., van Oorschot, P. C., & Vanstone, S. A. (1996). Handbook of Applied Cryptography. CRC Press.

2. Petitcolas, F. A. P., Anderson, R. J., & Kuhn, M. G. (1999). Information Hiding – A Survey. Proceedings of the IEEE, 87(7), 1062–1078.

3. Katz, J., & Lindell, Y. (2020). Introduction to Modern Cryptography (3rd ed.). CRC Press.

4. Johnson, N. F., Duric, Z., & Jajodia, S. (2001). Information Hiding: Steganography and Watermarking—Attacks and Countermeasures. Springer.

5. Mozilla Developer Network (MDN). Web Crypto API Documentation. *https://developer.mozilla.org/enUS/docs/Web/API/ Web_ Crypto_API*

6. HTML Living Standard. WHATWG HTML Specification. *https://html.spec.whatwg.org/*

7. W3Schools. CSS3 and HTML5 Tutorials. *https://www.w3schools.com/*

8. Python Software Foundation. Python 3.12.3 Documentation. *https://docs.python.org/3/*

9. Stallings, W. (2017). Cryptography and Network Security: Principles and Practice (7th ed.). Pearson Education.

10. Kumar, A., & Sharma, S. (2022). A Secure Image Steganography Technique Based on RSA and LSB. International Journal of Computer Applications, 184(5), 21-27