“ROBUST STEGANOGRAPHY ENCODER”

1. INTRODUCTION

In the present digital age, communication has become an essential part of everyday life. People share information through emails, messaging applications, social media platforms, and cloud services. While these technologies have made communication faster and easier, they have also introduced serious security and privacy challenges. Sensitive data such as passwords, confidential documents, academic information, and personal messages are at constant risk of interception and misuse.

Traditional security techniques such as encryption protect the content of the message by converting it into an unreadable format. However, encryption does not hide the existence of communication. An encrypted message clearly indicates that sensitive data is being transmitted, which may attract attackers or surveillance systems.

To overcome this limitation, a more subtle and effective technique called steganography is used. Steganography involves hiding secret information inside another digital medium in such a way that the presence of the hidden data is not noticeable. Images are one of the most commonly used carriers because small changes in pixel values are imperceptible to the human eye.

This project, titled “Robust Steganography Encoder”, focuses on building a full-stack web application that securely hides secret text messages inside images using Least Significant Bit (LSB) steganography. The application ensures that the visual quality of the image remains almost unchanged and verifies robustness using Peak Signal-to-Noise Ratio (PSNR) and compression simulation.

2. PROBLEM STATEMENT

With the increase in digital communication, ensuring the confidentiality and privacy of information has become a major concern. Although encryption techniques protect the message content, they still expose the existence of communication. This can raise suspicion and make the data vulnerable to attacks.

Moreover, images shared on social media or messaging platforms are often compressed automatically. Traditional steganography techniques fail under such transformations, leading to loss of hidden information.

The major problems addressed in this project are:

1. Insecure transmission of sensitive information
2. Visibility of encrypted messages
3. Loss of hidden data after image compression
4. Degradation of image quality after data embedding

3. PROPOSED SOLUTION

The proposed solution is a **Robust Steganography Encoder**, a web-based application that hides secret text inside image pixels using LSB steganography. The system is designed to be secure, robust, and user-friendly.

Key highlights of the solution:

- Uses image steganography instead of plain text encryption
- Maintains high visual quality of images
- Allows optional password-based encryption
- Verifies robustness using PSNR
- Simulates JPEG compression to test message survival

4. OBJECTIVES OF THE PROJECT

The primary objectives of this project are:

1. To design a secure method for hiding text inside digital images
2. To ensure minimal distortion in the encoded image
3. To allow accurate decoding of hidden messages
4. To simulate real-world compression scenarios
5. To measure image quality using PSNR
6. To provide a simple and interactive user interface
7. To demonstrate a real-world application of steganography

5. SCOPE OF THE PROJECT

The scope of this project is limited to **text-based steganography using images**. It includes:

- Encoding and decoding text messages in images
- JPEG and PNG image formats
- Browser-based frontend
- Server-side image processing

Out of Scope:

- Audio and video steganography
- Mobile application development
- Blockchain or cloud-based storage
- Large-scale distributed systems

6. TECHNOLOGY STACK

6.1 Frontend Technologies

React.js

- React is used to build the user interface of the application. It provides a component-based architecture, making the code modular, reusable, and easy to maintain. React also improves user experience by updating the UI dynamically without reloading the page.

Axios

- Axios is used to send HTTP requests from the frontend to the backend APIs. It simplifies data exchange and supports asynchronous operations.

Tailwind CSS / CSS

- Tailwind CSS or simple CSS is used to create a modern, clean, and responsive user interface.

6.2 Backend Technologies

Node.js

- Node.js is used as the backend runtime environment. It allows fast execution of JavaScript code and supports asynchronous processing.

Express.js

- Express.js is a lightweight web framework used to build RESTful APIs. It handles routing, middleware integration, and server logic.

6.3 Image Processing Library

Jimp

Jimp is a JavaScript image processing library used to:

- Read image pixels
- Modify RGB values
- Embed binary data
- Apply JPEG compression
- Save processed images

6.4 File Upload Handling

- **Multer**
- Multer is middleware used to handle multipart/form-data for image uploads. It stores uploaded images securely on the server.

7. SYSTEM ARCHITECTURE

The system follows a **Client-Server Architecture**.

Components:

1. User
2. Frontend (React)
3. Backend (Node.js + Express)
4. Image Processing Module (Jimp)
5. Storage (Uploads directory)

Architecture Flow:

1. User interacts with frontend
2. Frontend sends API requests
3. Backend processes image and message
4. Encoded image and results returned
5. Frontend displays output

8. FUNCTIONAL REQUIREMENTS

Encoding Module:

- Upload image
- Enter secret message
- Optional password encryption
- Embed message into image
- Download encoded image

Decoding Module:

- Upload encoded image
- Extract hidden message
- Display decoded text

Compression Module:

- Simulate JPEG compression
- Test decoding robustness

Quality Evaluation:

- Calculate PSNR
- Display PSNR value

9. NON-FUNCTIONAL REQUIREMENTS

1. High usability
2. Fast processing time
3. Minimal image distortion
4. Secure handling of data

5. Scalability for multiple users

10. WORKING METHODOLOGY

10.1 Encoding Process

1. User uploads a PNG or JPEG image
2. The image is received by the backend using Multer
3. Jimp reads the image pixel data
4. The secret message is converted into binary format
5. A delimiter is appended to mark the end of the message
6. Binary bits are embedded into the LSB of RGB channels
7. The encoded image is saved on the server
8. PSNR is calculated between original and encoded images
9. Encoded image and PSNR value are sent to the frontend

10.2 Decoding Process

1. User uploads the encoded image
2. Backend extracts LSB bits from pixels
3. Binary data is reconstructed
4. Extraction stops when delimiter is found
5. Binary data is converted back to text
6. Decoded message is returned to frontend

10.3 JPEG Compression Simulation

1. Encoded image is compressed using Jimp
2. Compression quality is set to 50%
3. Decoding is attempted again
4. Success or failure is recorded

11. LEAST SIGNIFICANT BIT (LSB) STEGANOGRAPHY

LSB steganography modifies the least significant bit of pixel values. Since the LSB contributes very little to the overall color value, changes are visually imperceptible.

Example:

Original Pixel Value: 11001010

Modified Pixel Value: 11001011

The difference is not noticeable to the human eye.

12. PSNR (PEAK SIGNAL-TO-NOISE RATIO)

PSNR is used to measure image quality after encoding.

Formula:

$PSNR = 10 \times \log10 (255^2 / MSE)$

- Higher PSNR indicates better image quality
- Values above 40 dB indicate excellent quality

13. SECURITY FEATURES

1. Hidden communication
2. Optional password-based encryption
3. No visible artifacts
4. Safe file handling

14. TESTING AND RESULTS

Test Cases:

- Encoding with short message
- Encoding with long message
- Decoding before compression
- Decoding after compression

Results:

- Image quality preserved
- Message successfully recovered
- Robustness verified

15. APPLICATIONS

1. Secure communication
2. Digital watermarking
3. Journalism
4. Academic data sharing
5. Military and defense communication

16. LIMITATIONS

1. Limited message size
2. Works only for images
3. Heavy compression may destroy data

17. FUTURE ENHANCEMENTS

1. Audio and video steganography
2. AI-based embedding
3. Mobile application
4. Cloud integration
5. Enhanced encryption

18. CONCLUSION

The **Robust Steganography Encoder** demonstrates an effective and practical approach to secure communication using image steganography. By combining frontend and backend technologies with image processing techniques, the system successfully hides secret messages while maintaining image quality and robustness.

This project highlights the importance of privacy, security, and robustness in modern digital communication systems.

19. REFERENCES

1. Gonzalez, Digital Image Processing
2. Node.js Official Documentation
3. React Official Documentation
4. Jimp Library Documentation