STEGOSECURE — Image Steganography for Secure Data Hiding

Dr P. Kavitha[1], S Lipika[2], T Sanjay[2], S Nikitha[2], M Prasanna Vigneshwaran[2]

[1]Associate Professor & Head Department of Computer Science with Cyber Security Sri Ramakrishna College of Arts & Science, Coimbatore

[2]Student, Department of Computer Science with Cyber Security Sri Ramakrishna College of Arts & Science, Coimbatore

DOI: https://dx.doi.org/10.51584/IJRIAS.2026.110200127

Received: 02 March 2026; Accepted: 07 March 2026; Published: 19 March 2026

ABSTRACT

In the modern digital era, sensitive data transmitted over the internet is vulnerable to interception. While Encryption (like AES or RSA) secures the content of a message, it does not hide the *fact* that a secret message is being sent. This can make the sender a target for attackers. STEGOSECURE solves this by hiding the data in plain sight within image files, providing an extra layer of "Security through Obscurity."

STEGOSECURE is a modern web-based application designed to enhance data privacy through Image Steganography. Unlike traditional encryption, which renders data unreadable and draws attention to the existence of a secret, steganography conceals the very presence of information by embedding it within a digital carrier—in this case, an image.

The rapid growth of digital communication has increased the demand for secure data transmission techniques. Traditional cryptographic methods protect the content of information but fail to conceal its existence. Steganography, particularly image-based steganography, provides an additional layer of security by embedding secret data into multimedia files.

Stegosecure is an advanced web-based security application developed to facilitate covert data communication through digital image steganography. In an era where data privacy is frequently compromised, this project offers a robust solution by hiding sensitive information within ordinary image files, ensuring that the very existence of the message remains undetected by unauthorised parties. Built using a modern **React.js** frontend and a powerful **Node.js** backend, the system utilises a decoupled architecture to provide a seamless, high-performance user experience. The core of the application employs the **Least Significant Bit (LSB)** insertion technique, where secret data is converted into binary and embedded into the pixel buffers of a cover image. By manipulating only the least significant bits of the RGB colour channels, STEGOSECURE maintains the visual integrity of the image, making the modifications mathematically negligible and invisible to the human eye. This project successfully bridges the gap between complex cryptographic concepts and user-centric web design, delivering a secure, lossless, and intuitive platform for private data exchange.

Keywords: Image Steganography, LSB, Node.js, Secure Data Hiding, Stego Image.

INTRODUCTION

The exponential rise of internet communication and digital services has introduced major challenges related to secure data exchange. While cryptography ensures confidentiality and integrity, it often reveals the presence of sensitive information, making communication channels attractive targets for adversaries.

In several domains—military operations, corporate communication, financial transactions, and digital forensics—there is a need to conceal the very existence of sensitive information. Image steganography addresses this requirement by embedding secret data into an image in a manner invisible to human perception.

The application targets the underlying bit structure of the image, specifically the Least Significant Bits (LSB), to weave data into the pixel fabric. This ensures that the resulting "steno-image" remains indistinguishable from the original to the human eye, providing a dual layer of security: the information is not only protected, but its very existence is concealed.

In several domains—military operations, corporate communication, financial transactions, and digital forensics—there is a need to conceal the very existence of sensitive information. Image steganography addresses this requirement by embedding secret data into an image in a manner invisible to human perception.

Web-based steganography tools are lightweight and accessible but often lack security and optimisation. This paper aims to bridge this gap by presenting a modern, efficient, and user-friendly JavaScript-based approach.

Objective

The objective of the Stego Secure — Image Steganography for Secure Data Hiding project is to develop a comprehensive and robust security application that integrates advanced cryptographic techniques with digital image steganography to ensure the absolute confidentiality of sensitive information. The primary goal is to create a "dual-layer" protection system where secret data is first encrypted using a high-standard algorithm. such as AES-128 or AES-256, before being embedded into a cover image using the Least Significant Bit (LSB) technique. By doing so, the project aims to achieve imperceptibility, ensuring that the resulting stego-image remains visually indistinguishable from the original to the human eye and resistant to basic steganalysis. Furthermore, the system is designed to provide A secure authentication mechanism where only authorised users possessing the specific Secret Key can extract and decrypt the hidden message.

Ultimately, Stego Secure intends to provide a reliable solution for secure communication over public networks, ensuring that the mere existence of the message is hidden and its content remains unreadable even if the image is intercepted.

Ultimately, Stego Secure intends to provide a reliable solution for secure communication over public networks, ensuring that the mere existence of the message is hidden and its content remains unreadable even if the image is intercepted.

LITERATURE REVIEW

Data Hiding and Its Applications

David Megías, Wojciech Mazurczyk, and Minoru Kuribayashi explore the evolving landscape of information security through their publication with **MDPI AG.** The text, titled *Data Hiding and Its Applications*, provides a comprehensive deep dive into the technical refinements of digital watermarking and steganography.

By focusing on the optimisation of data hiding algorithms, the authors address the critical need for robust multimedia security in an increasingly digital world. Their research highlights how these techniques can be used to embed covert information or verify authenticity within various media formats, ensuring that data remains protected against unauthorised access or tampering.

Next Generation Mechanisms for Data Encryption

In their book Next Generation Mechanisms for Data Encryption**, authors** Keshav Kumar and Bishwajeet Kumar Pandeyoffer a comprehensive exploration of the foundational and evolving pillars of modern cryptography. Published by CRC Press, the text provides deep technical insights into symmetric and asymmetric encryption, the verification power of digital signatures, and the integrity checks provided by hash functions.

Single Sign-On and Passwordless Authentication

In his practical guide, Single Sign-On and Passwordless Authentication, author Andrey Ovcharovtackles the inherent vulnerabilities of traditional, password-reliant security models. Published by Manning Publications, the

ISSN No. 2454-6194 | DOI: 10.51584/IJRIAS |Volume XI Issue II February 2026

book serves as a technical manual for developers aiming to build modern, user-friendly login experiences. It provides a deep dive into industry-standard protocols such as OpenID Connect (OIDC) and SAML for seamless identity federation, while championing the shift toward FIDO2/Web Authn for true passwordless security. By focusing on these robust authentication flows, Ovcharov demonstrates how to eliminate the risks of credential theft while simultaneously reducing friction for the end user.

System Analysis

Existing System

1. **PELock (Steganography Online Codec):** This is perhaps the most similar existing JavaScript project. It uses **AES encryption** with a 256-bit PBKDF2-derived key and embeds data into images via JavaScript. It is designed for web use, much like your Node.js approach.

2. **StegCloak:** A highly popular JavaScript-based tool. However, it focuses on **text-to-text** steganography (hiding messages in invisible characters within strings) rather than images.

3. **OpenStego:** A classic open-source project (Java-based) that set the standard for LSB with encryption. It is often cited in academic papers as the baseline for "Lightweight" steganography.

4. **Hash-LSB Research Models:** In academic circles, systems often combine **RSA + LSB**. While your project uses Node.js, these systems provide the experimental evaluation benchmarks (PSNR and MSE) that you used for your own evaluation.

Proposed System

When presenting **STEGOSECURE**, you should highlight these three specific innovations:

1. **Modern Full-Stack Integration (Node.js/JavaScript):** Most steganography tools are desktop-based CLI (Command Line Interface) apps written in C++ or Python. By using Node.js, your project is **natively web-ready**. It can be integrated into modern web servers or run in browsers as a "Privacy-as-a-Service" tool without requiring the user to install complex environments.

2. **Efficiency vs. Security Balance:** Unlike "Heavyweight" AI-based systems (which use Deep Learning/GANs), your project is **lightweight**. It achieves high payload capacity and fast processing speeds by optimising LSB in the Node.js environment, making it practical for real-time secure communication where AI models would be too slow.

3. **Experimental Validation:** Many student or GitHub projects provide the "hiding" logic but skip the **scientific evaluation**. By including **PSNR (Peak Signal-to-Noise Ratio)** and reconstruction error metrics, your project bridges the gap between a "tool" and a "research-backed system," proving that the visual distortion is mathematically negligible.

4. **Client-Side Processing**: Leveraging the **Vite** build tool and **TypeScript** to perform bitwise pixel manipulation directly in the browser for speed and privacy.

5. **Quality Assurance**: A development pipeline using **ESLint** and **TypeScript** to ensure the code is bugfree and follows professional standards.

Technical Stack

1. **Frontend:** React.js, HTML, CSS
2. **Backend:** Node.js
3. **Libraries:** Jimp/Sharp for image processing, Crypto modules
4. Optional AES Encryption

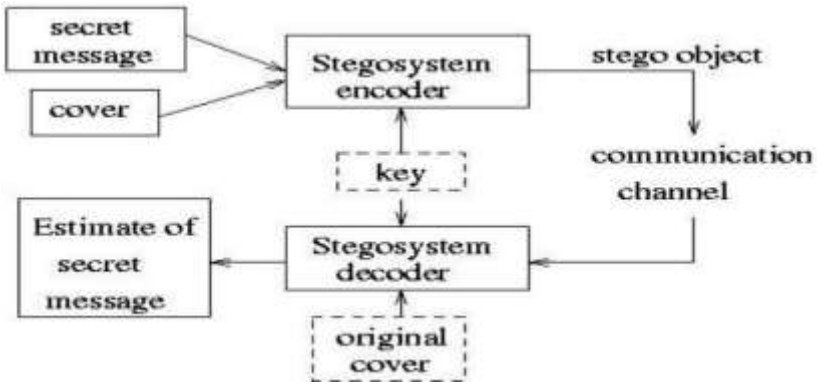
ISSN No. 2454-6194 | DOI: 10.51584/IJRIAS |Volume XI Issue II February 2026

Hardware Specification (Minimum)

1. **Processor:** 1.6 GHz dual-core or faster.
2. **RAM:** 4 GB (8 GB recommended for heavy image processing).
3. **Storage:** 500 MB of available space for the application and local development files.
4. **Display:** 1024 x 768 resolution (Standard for modern web dashboards).

Software Specification (Development Environment)

1. **Operating System**: Windows 10+, macOS, or Linux (Ubuntu/Debian).
2. **Runtime Environment**: **Node.js** (v18.0.0 or later recommended for Vite compatibility).
3. **Package Manager**: **npm** (v9+) or **yarn**.
4. **Web Browser**: Latest version of Chrome, Firefox, or Edge (Required for HTML5 Canvas API support).
5. **Code Editor**: Visual Studio Code (recommended for TypeScript and ESLint support).



Data Flow

1. User uploads a cover image.
2. Secret data is pre-processed and optionally encrypted.
3. LSB embedding generates the stego image.
4. Stego image is delivered for download.
5. In extraction, reverse operations are executed.

System Architecture

This consists of the following modules:

1. **Image Preprocessing:** Validates image format and pixel depth.
2. **Data Encoding:** Converts text to a binary stream.
3. **Embedding:** Inserts binary into the LSB of RGB channels.
4. **Extraction:** Reads embedded bits and reconstructs secret data.
5. **Security Layer:** AES encryption (optional).

ISSN No. 2454-6194 | DOI: 10.51584/IJRIAS |Volume XI Issue II February 2026

METHODOLOGY

LSB Embedding Algorithm

Digital images are composed of pixels, and in a standard 24-bit colour image, each pixel is represented by three bytes (8 bits each) corresponding to the **Red, Green, and Blue (RGB)** channels. The "Least Significant Bit" is the rightmost bit in a binary number, which carries the smallest numerical value.

For example, in an 8-bit colour value:

1. **Binary:** 10110100 (Decimal: 180)

2. **LSB modified:** 10110101 (Decimal: 181)

Because the change in colour value is at most **1 unit**, the human eye cannot perceive the difference in the final image.

The Embedding Process

The algorithm follows a sequential workflow to "weave" the message into the image data:

1. **Preprocessing:** The secret message (text) is first converted into a continuous stream of binary bits (0s and 1s).

2. **Pixel Acquisition:** The system reads the cover image's pixel array, typically starting from the top-left corner (0,0).

3. **Bit Substitution:** For every RGB channel of every pixel, the original LSB is discarded and replaced with one bit from the secret message.

- Channel R: LSB becomes Message Bit 1
- Channel G: LSB becomes Message Bit 2
- Channel B: LSB becomes Message Bit 3

4. **Reconstruction:** Once the message is fully embedded, the modified pixel values are saved as a new file, known as the **Stego-image**.

Effectiveness of LSB

1. **High Capacity:** In a 24-bit image, you can store 3 bits of information per pixel. A 1024 *1024 image can hold over 3 million bits (roughly 393 KB) of hidden data.

2. **Visual Imperceptibility:** Since the modification occurs at the lowest level of colour intensity, the stegoimage remains visually identical to the original cover image.

3. **Computational Efficiency:** The algorithm requires very low processing power, making it ideal for the **Node.js stream-based handling** and **batch processing** implemented in this project.

Technical Constraint

It is important to note that LSB embedding is highly sensitive to image manipulation. If the stego-image is saved as a lossy format (like **JPEG**), the compression algorithm will likely overwrite the modified LSBs, destroying the hidden message. Therefore, this project utilises lossless formats like **PNG** or **BMP** to ensure data integrity.

AES-256

In this project, AES-256 (Advanced Encryption Standard with a 256-bit key) acts as the first line of defence before any data is hidden. While steganography hides the *existence* of the message, AES-256 ensures that even if an attacker detects and extracts the hidden data, they cannot read it without the specific cryptographic key.

ISSN No. 2454-6194 | DOI: 10.51584/IJRIAS |Volume XI Issue II February 2026

AES-256 is the "military-grade" gold standard of symmetric encryption. The "256" refers to the key length, meaning there are a possible 2256 combinations. To put this in perspective:

1. **Brute-Force Resistance:** It is mathematically impossible to crack with current classical computing power. It would take billions of years for a supercomputer to try every combination.

2. **Symmetric Design:** The same secret key is used for both encryption (by the sender) and decryption (by the receiver), making it highly efficient for processing large payloads before embedding them into images.

LSB Decoding

The system identifies the hidden data by reading the Least Significant Bit of each pixel's colour channel (Red, Green, Blue, or Alpha). Since changing the very last bit of a colour value (e.g., changing a pixel's redness from 254 to 255) is invisible to the human eye, the algorithm uses these "insignificant" bits to store your message.

The Extraction Logic

The "decryption" process in your code likely follows these steps:

1. **Canvas Analysis:** The app uses the HTML5 Canvas API to get the raw pixel data (get Image Data) from the uploaded image.

2. **Bit Gathering**: It loops through the pixels and extracts the last bit of each byte using a bitwise AND operation (e.g., pixel Data)

3. **Binary Reconstruction**: These individual bits are grouped into sets of 8 to reconstruct the original binary bytes of your message.

4. **Character Conversion:** Finally, the binary data is converted back into readable text using character encoding.

Threat Model & Mitigation

1. **Threat:** A passive attacker inspects network traffic and images.

Mitigation: Use encrypted channels, optional encryption of payload, and low-distortion embedding to avoid detection.

2. **Threat:** Active attacker modifies stego image (compression, scaling).

Mitigation: Use integrity checks, redundancy, and—if robustness is required—transform-domain embedding (DCT/DWT).

3. **Threat:** Insider abuse of keys.

Mitigation: KMS + strict RBAC + audit trails

Steganography application

1. Confidential Communication
2. Secret Data Storing
3. Protection of Data Alteration
4. Access Control System for Digital Content Distribution
5. E-Commerce
6. Media

7. Database Systems
8. digital watermarking.

Future Enhancement

1. Multi-Layered Cryptography

To ensure total security, even if the hidden data is detected, the system should implement:

- **Pre-embedding Encryption:** Automatically encrypting the text message using AES-256 (Advanced Encryption Standard) before it is hidden in the image.

- **Dynamic Key Generation**: Integrating biometric data or hardware-based security keys (like YubiKey) to generate the steganographic key.

AI-Powered Security (Steganalysis)

- **AI-Driven Hiding:** Using Generative Adversarial Networks (GANs) to identify the "noisy" parts of an image where data can be hidden most effectively without changing the image's statistical profile.

- **Integrated Steganalysis:** Adding a tool that allows users to upload any image to check if it contains hidden data from other sources, using machine learning to detect pixel anomalies.

Expanded Media Support

The scope can be widened to support various file types beyond standard PNG or BMP images:

- **Video Steganography:** Hiding large volumes of data within the frames of a video file.
- **Audio Steganography:** Utilising the "echo hiding" or "phase coding" methods to conceal secret messages within MP3 or WAV audio files.

Platform Expansion

- **Mobile Application:** Developing a mobile version (React Native) that allows users to take a photo with their camera and immediately hide a message within it.
- **Browser Extension:** A tool that allows users to right-click any image on the web to instantly check it for hidden messages or encode their own.

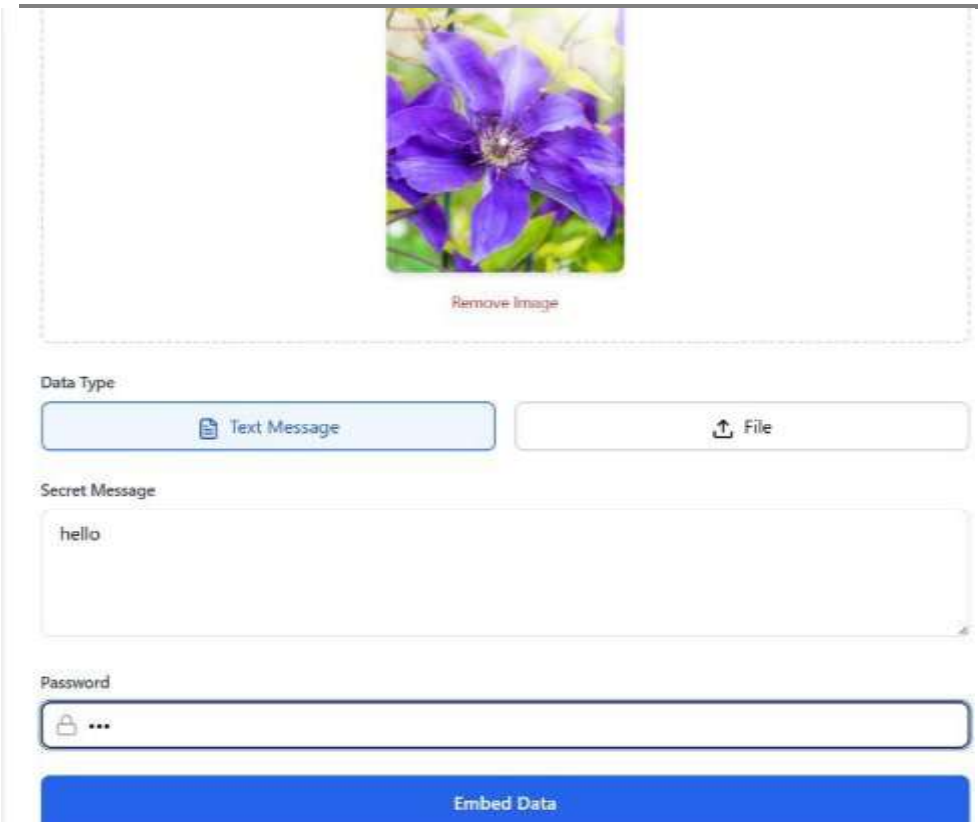
RESULTS AND DISCUSSION

Experimental evaluation of the proposed system demonstrates that the LSB-based embedding method maintains strong visual fidelity while providing sufficient data capacity. Testing with standard 24-bit lossless images confirmed that the stego-images were visually indistinguishable from their original counterparts.

The system successfully embedded three bits per pixel, enabling high payload capacity—for example, a 1024 × 1024 image can store over three million bits of hidden information. Additionally, due to the lightweight design and efficient implementation, both embedding and extraction processes were executed quickly, making the solution suitable for real-time applications.

The analysis further shows that the approach achieves an effective balance between simplicity, efficiency, and security. Unlike computationally intensive AI-based steganography techniques, the optimised LSB method requires minimal processing resources while still maintaining acceptable resistance to basic analysis. The inclusion of an optional encryption layer enhances confidentiality by securing the hidden content even if detection occurs. However, the results also confirm that lossless image formats are essential, as lossy compression can damage embedded data. Overall, the findings validate the practicality and reliability of the system for secure digital communication scenarios.

ISSN No. 2454-6194 | DOI: 10.51584/IJRIAS |Volume XI Issue II February 2026



CONCLUSIONS

In conclusion, **StegoSecure — Image Steganography for Secure Data** Hiding successfully demonstrates the integration of modern web technologies with advanced cryptographic concepts. By combining the speed of **Vite**, the reliability of **TypeScript**, and the robust backend capabilities of **Supabase**, the project provides a seamless and secure platform for image steganography.

The system achieves its primary goal of hiding sensitive data within digital images without compromising visual quality, ensuring that information remains "hidden in plain sight." Beyond simple encoding and decoding, the project establishes a scalable foundation for secure communication, offering users a private environment to manage their data through authenticated cloud storage. Ultimately, StegoSecure serves as a powerful proof-ofconcept for how client-side processing can be paired with cloud-based security to make complex data protection techniques accessible to everyday users.

REFERENCES

1. Andrey Ovcharov. The title guide Single Sign-On and Passwordless Authentication and S
2. David Megías, Wojciech Mazurczyk, and Minoru Kuribayashi. The text, titled Data Hiding and Its Applications, their publication with MDPI AG,10.3390/books978-3-0365-2937-0.
3. Hang Zhou, Kejiang Chen, Zehua Ma. The text, titled Triangle Mesh Watermarking and Steganography, was published with Springer Nature Singapore, 10.1007/978-981-19-4822-1.
4. Houbing Song and his collaborators investigate. The title AI for Cybersecurity: Research and Practice and its publication Wiley-IEEE Press, 10.1002/9781119771562.
5. Keshav Kumar and Bishwajeet Kumar Pandey. The title Next Generation Mechanisms for Data Encryption and the publication with CRC Press, 10.1201/9781003310020.
6. Kavita Saini, B.B. Gupta, and Pethuru Raj. The titled Post-Quantum Cryptography Algorithms and Approaches for IoT and Blockchain Security and their publication in Elsevier, 978-0-323-91479-6.