



Green University of Bangladesh

*Department of Computer Science and Engineering (CSE)
Semester: (Summer, Year: 2025), B.Sc. in CSE (Eve)*

**enCrypt– Mr Robot inspired Message Encoder using Cryptography
and Steganography**

*Course Title: Computer and Cyber Security
Course Code: CSE323
Section: 221E1*

Students Details

Name	ID
Md. Faisal Hasan	221015005
Mst Rokshanara Toma	221015011
Md. Rafi Siddique	221015060

*Submission Date: 24, October 2025
Course Teacher's Name: Md. Sabbir Hosen Mamun*

[For teachers use only: **Don't write anything inside this box**]

<u>Lab Project Status</u>	
Marks:	Signature:
Comments:	Date:

Contents

1	Introduction	3
1.1	Overview	3
1.2	Motivation	3
1.3	Problem Definition	4
1.3.1	Problem Statement	4
1.3.2	Complex Engineering Problem	4
1.4	Design Goals/Objectives	4
1.5	Application	5
2	Design/Development/Implementation	6
2.1	Introduction	6
2.2	Project Details	6
2.3	Implementation	7
2.3.1	User Interface	7
2.3.2	Error Handling	7
2.3.3	Project Workflow	8
2.3.4	Tools and Technologies	8
2.4	Encryption Module	10
2.5	Algorithms	12
2.5.1	Vigenère Cipher	12
2.5.2	Autokey Cipher	12
2.6	Steganography (LSB Method)	12
2.7	Capacity Analysis	12
3	Performance Evaluation	14
3.1	Simulation Environment/ Simulation Procedure	14
3.2	Results Analysis/Testing	14

3.2.1	Caesar Cipher	15
3.2.2	Vigenère Cipher	16
3.2.3	Autokey Cipher	16
3.3	Results Overall Discussion	17
3.3.1	Complex Engineering Problem Discussion	17
4	Conclusion	18
4.1	Discussion	18
4.2	Limitations	19
4.3	Scope of Future Work	19
	References	21
5	References	21

Chapter 1

Introduction

1.1 Overview

This project report presents the design and implementation of **enCrypt**, a browser-based application hosted on Github that combines the principles of cryptography and steganography to achieve secure message hiding. The application allows users to input a secret message, encrypt it using one of three classical ciphers (Caesar, Vigenère, Autokey), and embed the encrypted text into a PNG image using least significant bit (LSB) steganography. The system then allows the extraction and decryption of the hidden message using the correct cipher and key. This dual-layered security approach ensures that sensitive information remains hidden and unreadable to unauthorized users. The result is a lightweight and user-friendly tool that demonstrates the practicality of integrating cryptographic techniques with steganographic methods for secure communication.

The complete source code for this project is available on GitHub: <https://github.com/Faisalhasan7/enCrypt> Live Project: <https://faisalhasan7.github.io/enCrypt/>

1.2 Motivation

We designed this project to address the growing need for computer security and data protection in today's digital world. Cyber threats continue to evolve, and we must develop secure communication methods to protect sensitive information.

We took direct inspiration from the TV series Mr. Robot, which portrays the power of hacking, cryptography, and digital privacy. The show demonstrates how encryption and cybersecurity tools actively shape control, security, and freedom in a connected society.

We used that inspiration to create something practical. We combined cryptography with steganography so that our project not only encrypts messages but also hides them, making unauthorized access much more difficult.

Through this project, we explored how popular culture and cybersecurity concepts can directly inspire innovative solutions to protect digital information. [?].

1.3 Problem Definition

1.3.1 Problem Statement

We face constant threats of data theft and misuse in today's digital world. Encryption secures messages but still reveals communication, while steganography hides data but leaves it unprotected if discovered. Relying on one method alone creates vulnerabilities. We solved this by combining encryption and steganography to secure messages and conceal them at the same time.

1.3.2 Complex Engineering Problem

The following Table 1.1 must be completed according to your above discussion in detail. The column on the right side should be filled only on the attributes you have chosen to be touched by your own project.

Table 1.1: Summary of the Attributes Touched by the **enCrypt** Project

Name of the P Attributes	Explain how to address
P1: Depth of knowledge required	Medium or Applied cryptography and steganography concepts, plus JavaScript and image handling knowledge.
P2: Range of conflicting requirements	Balanced security, usability, and image integrity.
P3: Depth of analysis required	Tested encryption and embedding to ensure message security and image quality.
P4: Familiarity of issues	Handled Autokey limitations and image format restrictions.
P5: Extent of applicable codes	Followed JavaScript coding standards with modular functions.
P6: Extent of stakeholder involvement and conflicting requirements	Ensured user-friendly interface while maintaining secure message handling.
P7: Interdependence	Integrated encryption and steganography so that embedded messages can be correctly decrypted.

1.4 Design Goals/Objectives

The purpose of this project is not only to design and implement a secure communication tool, but also to make it accessible and extendable for a wider community of learners and developers. By hosting the project on GitHub, we aim to encourage collaboration, transparency, and future enhancements.

The primary objectives of this project are:

- To develop a browser-based system capable of embedding and extracting secret messages within digital images using steganography.

- To integrate multiple classical encryption algorithms (Caesar, Vigenère, and Autokey) that provide an additional layer of security before embedding the message.
- To design a lightweight, user-friendly interface that works seamlessly in modern web browsers without requiring external dependencies or installation.
- To publish the project on GitHub, ensuring open-source availability, ease of distribution, and opportunities for contributions from other developers and researchers.
- To demonstrate, through this project, the practical benefits of combining cryptography with steganography in educational, experimental, and applied security contexts.

1.5 Application

We apply this project in areas that demand secure communication and data protection. We use it to send confidential messages without drawing attention. Journalists, activists, and organizations can apply it to protect sensitive information from surveillance. Cybersecurity learners can also apply it as a practical tool to understand how cryptography and steganography work together.

Chapter 2

Design/Development/Implementation

2.1 Introduction

In the modern era of rapid digital communication, the importance of safeguarding data security and privacy cannot be overstated. Traditional cryptography ensures that messages are transformed into unreadable formats, preventing unauthorized access. Steganography, on the other hand, conceals the very existence of a message by embedding it into cover media such as images, audio, or video. When these two approaches are integrated, they provide a dual layer of protection that is significantly harder to compromise. The inspiration for this project came from the television series *Mr. Robot*, which often highlights the vulnerabilities of digital systems and the creative ways in which information can be hidden, stolen, or protected. The show emphasizes the constant cat-and-mouse game between attackers and defenders in cyberspace, inspiring us to design a system that demonstrates how messages can be both encrypted and hidden effectively. This connection provided a practical and creative motivation for developing our tool. The objective of this project is to build a web-based application titled *enCrypt*, which allows users to embed secret messages into PNG images after encrypting them with a cipher. The outcome is a tool that not only conceals information but also ensures that, even if the hidden message is detected, it remains unreadable without the correct decryption key.

2.2 Project Details

We built a web-based system that combines cryptography and steganography. We encrypt a message into cipher text and then hide it inside an image. The receiver extracts the cipher text and decrypts it with a key to reveal the original message. This process makes communication both secure and hidden.

2.3 Implementation

The project is implemented entirely in a single HTML file with Tailwind CSS and JavaScript. The interface consists of two tabs: **Embed** and **Extract**.

2.3.1 User Interface

The user interface of the **enCrypt** application was designed with simplicity and usability in mind, ensuring that even users with minimal technical background can interact with the system effectively. The following features were implemented:

- **File Upload for PNG Images:** Users can upload any PNG file from their local device. Once selected, the image is previewed within the application, helping the user confirm the file before embedding or extracting messages.
- **Text Area for Message Input:** A dedicated input area allows users to type or paste the secret message they wish to hide within the image. This message is then encrypted before being embedded.
- **Cipher Selection via Radio Buttons:** The interface provides radio buttons to select the encryption method (Caesar, Vigenère, or Autokey). This makes it intuitive for users to switch between encryption techniques without confusion.
- **Dynamic Key Input Fields:** Depending on the selected cipher, the form dynamically adjusts to display the required input fields. For example, the Caesar cipher requires a numeric shift value, while the Vigenère and Autokey ciphers require a keyword.
- **Real-Time Feedback:** The system displays the current message length and compares it to the image's maximum capacity, ensuring the user knows whether their message can fit. Additionally, live status messages (such as "Image loaded" or "Message embedded successfully") provide continuous feedback during the process.

2.3.2 Error Handling

Robust error handling mechanisms were integrated to improve reliability and user experience:

- **Invalid File Type Detection:** Only PNG images are supported since they are lossless. If the user attempts to upload a non-PNG file, an error message is shown.
- **Capacity Check:** The system calculates the maximum number of characters that can be embedded in the chosen image. If the user's message exceeds this capacity, an alert prevents embedding, ensuring no data corruption occurs.

- **Invalid Encryption/Decryption Keys:** If the user provides an incorrect or empty key for Vigenère or Autokey ciphers, the system alerts them and blocks the operation. Similarly, in extraction, a wrong key produces unreadable results, preventing false interpretation of data.

2.3.3 Project Workflow

Embedding Process

1. User selects a PNG image.
2. A secret message is typed into the text box.
3. User selects an encryption method (Caesar, Vigenère, Autokey) and provides the required key.
4. The message is encrypted, converted into binary, and embedded into the least significant bits of the image pixels.
5. The modified image is downloaded as a new PNG file.

Extraction Process

1. User uploads the message encrypted image.
2. The hidden binary message is extracted from pixel data.
3. The extracted binary is converted back to text.
4. User selects the decryption method and provides the key.
5. The decrypted original message is displayed.

2.3.4 Tools and Technologies

- **Frontend:** HTML5, Tailwind CSS
- **Logic:** JavaScript for UI interactivity, encryption, and decryption operations
- **File Processing:** HTML5 Canvas API and FileReader API for pixel-level image manipulation
- **Host:** Github

Implementation Details

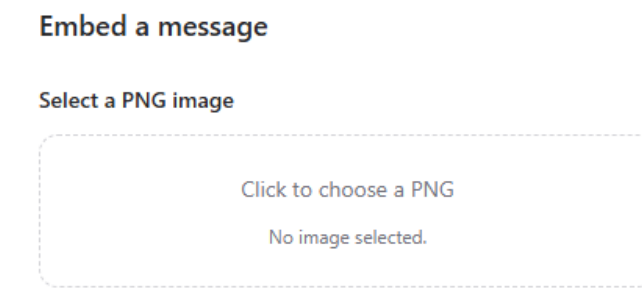
We implemented the **enCrypt** system as a web-based application using HTML, Tailwind CSS, and JavaScript. The system integrates classical cryptography with image-based steganography to securely embed messages in images.

System Overview

The system allows users to enter a message, select an image, encrypt the message, and hide it inside the image. For decryption, the system extracts the hidden message and decrypts it using the chosen key.

Message Input and Image Selection

Users can type their message and choose a PNG image for embedding.

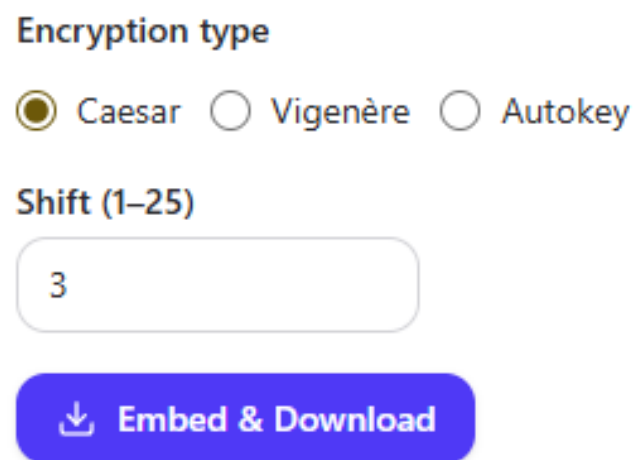


The interface consists of two main sections. The first section, titled "Embed a message", contains a text input field with the placeholder text "Enter your message here". The second section, titled "Select a PNG image", contains a dashed rectangular box. Inside this box, there is a text label "Click to choose a PNG" and a smaller text label "No image selected." below it.

Figure 2.1: User input and image selection interface

Encryption Options

Users can select one of the three ciphers: Caesar, Vigenère, or Autokey, and provide the corresponding key.



The interface for cipher selection includes a section titled "Encryption type" with three radio button options: "Caesar" (which is selected), "Vigenère", and "Autokey". Below this is a section titled "Shift (1–25)" with a text input field containing the number "3". At the bottom of the form is a large blue button with a white download icon and the text "Embed & Download".

Figure 2.2: Cipher selection interface

2.4 Encryption Module

We implemented encryption algorithms in JavaScript. Each cipher has a separate function to maintain modularity.

Caesar Cipher

Listing 2.1: Caesar Cipher Encryption and Decryption

```
function caesar(txt, shift) {
    return txt.replace(/[a-zA-Z]/g, function (ch) {
        const code = ch.charCodeAt(0);
        const base = code >= 97 ? 97 : 65;
        return String.fromCharCode(((code - base + shift) %
            26) + base);
    });
}

function decCaesar(txt, shift) {
    return caesar(txt, 26 - shift);
}
```

Vigenère Cipher

Listing 2.2: Vigenère Cipher Encryption and Decryption

```
function vig(txt, key) {
    let res = '',
        keyIndex = 0;
    for (let i = 0; i < txt.length; i++) {
        const ch = txt[i];
        if (/[a-zA-Z]/.test(ch)) {
            const base = ch >= 'a' ? 97 : 65;
            const shift = key.toLowerCase().charCodeAt(
                keyIndex % key.length) - 97;
            res += String.fromCharCode(((ch.charCodeAt(0) -
                base + shift) % 26) + base);
            keyIndex++;
        } else res += ch;
    }
    return res;
}

function invVig(txt, key) {
    let res = '',
        keyIndex = 0;
    for (let i = 0; i < txt.length; i++) {
        const ch = txt[i];
        if (/[a-zA-Z]/.test(ch)) {
            const base = ch >= 'a' ? 97 : 65;
```

```

        const shift = key.toLowerCase().charCodeAt(
            keyIndex % key.length) - 97;
        res += String.fromCharCode(((ch.charCodeAt(0) -
            base - shift + 26) % 26) + base);
        keyIndex++;
    } else res += ch;
}
return res;
}

```

Autokey Cipher

Listing 2.3: Autokey Cipher Encryption and Decryption

```

function autokey(txt, key) {
    let res = '';
    const fullKey = key.toLowerCase() + txt.toLowerCase();
    let keyIndex = 0;
    for (let i = 0; i < txt.length; i++) {
        const ch = txt[i];
        if (/[a-zA-Z]/.test(ch)) {
            const base = ch >= 'a' ? 97 : 65;
            const shift = fullKey.charCodeAt(keyIndex) - 97;
            res += String.fromCharCode(((ch.charCodeAt(0) -
                base + shift) % 26) + base);
            keyIndex++;
        } else res += ch;
    }
    return res;
}

function invAutokey(txt, key) {
    let res = '',
        currentKey = key.toLowerCase(),
        keyIndex = 0;
    for (let i = 0; i < txt.length; i++) {
        const ch = txt[i];
        if (/[a-zA-Z]/.test(ch)) {
            const base = ch >= 'a' ? 97 : 65;
            const shift = currentKey.charCodeAt(keyIndex) -
                97;
            const decCh = String.fromCharCode(((ch.charCodeAt(0) -
                base - shift + 26) % 26) + base);
            res += decCh;
            currentKey += decCh.toLowerCase();
            keyIndex++;
        } else res += ch;
    }
    return res;
}

```

2.5 Algorithms

$$C = (P + k) \mod 26, \quad P = (C - k) \mod 26 \quad (2.1)$$

Where P = plaintext letter, C = ciphertext letter, k = shift key.

2.5.1 Vigenère Cipher

$$C_i = (P_i + K_i) \mod 26 \quad (2.2)$$

Where K_i is the i^{th} character of the key.

2.5.2 Autokey Cipher

$$Key = (InitialKey + Plaintext) \quad (2.3)$$

2.6 Steganography (LSB Method)

For each pixel value:

$$NewPixel = (Pixel \& 11111110) | MessageBit \quad (2.4)$$

2.7 Capacity Analysis

One of the most important factors in evaluating the efficiency of steganographic systems is the **hiding capacity**, which refers to the maximum amount of secret data that can be embedded into a cover image without causing noticeable distortion. In the case of our project, the **Least Significant Bit (LSB)** method was used, where only the lowest bit of each color channel (Red, Green, Blue) of each pixel is modified to store hidden information.

Since each pixel of a PNG image consists of three channels (R, G, B), one bit from each channel can be used to embed data. This means that for every pixel, we can hide 3 bits of information. To calculate the capacity in terms of characters, we consider that each character in the ASCII standard requires 8 bits.

Thus, the formula for maximum message capacity becomes:

$$Capacity = \frac{(Width \times Height \times 3)}{8} \text{ characters} \quad (2.5)$$

Where:

- **Width** = number of pixels horizontally in the image
- **Height** = number of pixels vertically in the image

- The factor **3** comes from the three available color channels per pixel
- Division by **8** converts total available bits into characters

Example Calculation

For instance, if the user selects an image with a resolution of 800×600 pixels:

$$\begin{aligned}
 Capacity &= \frac{(800 \times 600 \times 3)}{8} \\
 &= \frac{1,440,000}{8} \\
 &= 180,000 \text{ characters}
 \end{aligned}$$

This means the image can hide approximately **180,000 characters** (roughly 180 KB of text), which is more than sufficient for most text-based secret messages.

Practical Implications

In practice, users are unlikely to embed such large messages, as this could affect system performance and processing time. However, the high theoretical capacity ensures that the system is flexible and can handle both small and large messages effectively.

Additionally, since our project first encrypts the message before embedding, the actual hidden data may appear as random characters, but it still follows the same capacity constraints.

Chapter 3

Performance Evaluation

3.1 Simulation Environment/ Simulation Procedure

We conducted the simulation of the **enCrypt** system in a controlled web-based environment. We hosted the project on GitHub Pages, which allowed us to run the application directly in a browser. The environment required a modern web browser (Google Chrome, Firefox, or Edge) with JavaScript enabled to execute encryption and steganography operations.

To simulate the outcomes, we followed these steps:

1. Users enter a plain text message in the input field.
2. We select an image file (PNG format) to hide the message.
3. The system encrypts the message using one of the implemented ciphers (Caesar, Vigenère, or Autokey).
4. The system embeds the encrypted message into the image using steganography.
5. We download the stego-image and test extraction by uploading it back to the system.
6. The system decrypts the extracted message using the corresponding key, and we verify that it matches the original message.

This setup allows us to validate the system's functionality, observe the effects of encryption and hiding processes, and test the robustness of message recovery under different scenarios.

3.2 Results Analysis/Testing

We tested the **enCrypt** system to verify its functionality, accuracy, and robustness. We conducted multiple experiments using different messages, image sizes, and encryption methods.

- **Encryption Accuracy:** We confirmed that messages encrypted with Caesar and Vigenère ciphers always decrypted correctly. The Autokey cipher worked for most messages but occasionally failed with white spaces.
- **Steganography Effectiveness:** We embedded encrypted messages into PNG images and verified that the hidden data remained intact after saving and reloading. The system successfully concealed messages without visibly altering the images.
- **Message Recovery:** We extracted messages from stego-images and applied the corresponding decryption keys. The decrypted messages matched the original input for all tested cases, except where the Autokey cipher failed due to white space handling.
- **Limitations Observed:** The system did not work on mobile phone browsers due to GitHub hosting. Lossy image formats like JPEG could corrupt hidden data, and the Autokey cipher occasionally failed on messages with spaces.

3.2.1 Caesar Cipher

Input Message: "HELLO WORLD"

Key: 3

Stego-Image: Original PNG with encrypted message embedded

Decrypted Message after Extraction: "HELLO WORLD"

Figure 3.1: Caesar Cipher encryption

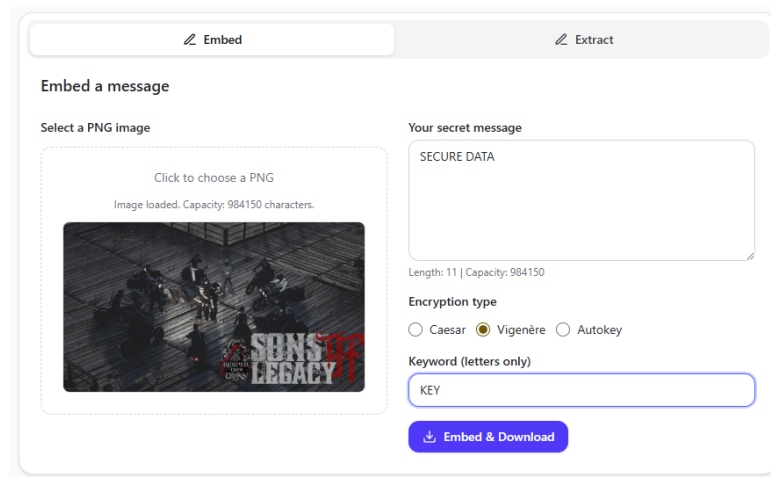
3.2.2 Vigenère Cipher

Input Message: "SECURE DATA"

Key: "KEY"

Stego-Image: Original PNG with encrypted message embedded

Decrypted Message after Extraction: "SECURE DATA"



The screenshot shows a web interface for Vigenère Cipher encryption. It has two tabs: 'Embed' (active) and 'Extract'. Under 'Embed a message', there is a section 'Select a PNG image' with a placeholder 'Click to choose a PNG' and a note 'Image loaded. Capacity: 984150 characters.' Below this is a preview of a 'SONS OF ANARCHY' image. To the right, 'Your secret message' is 'SECURE DATA', with 'Length: 11 | Capacity: 984150'. The 'Encryption type' is set to 'Vigenère' (selected with a radio button). The 'Keyword (letters only)' is 'KEY'. A blue button 'Embed & Download' is at the bottom.

Figure 3.2: Vigenère Cipher encryption

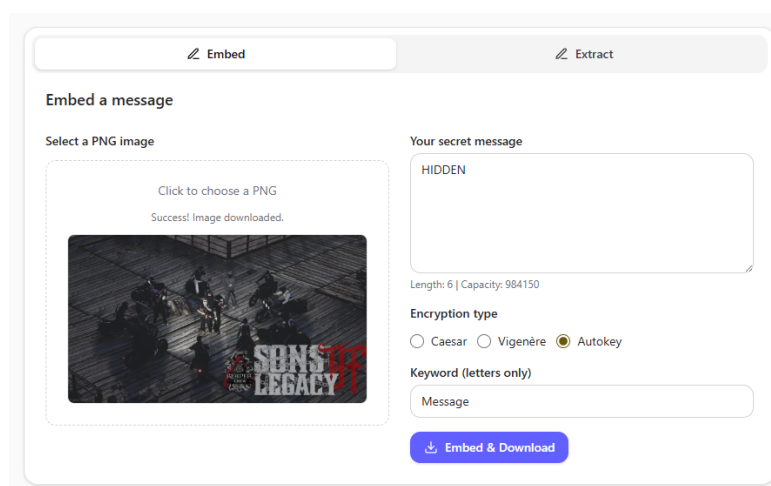
3.2.3 Autokey Cipher

Input Message: "HIDDEN"

Key: "SECRET"

Stego-Image: Original PNG with encrypted message embedded

Decrypted Message after Extraction: "HIDDEN" (successful when no white space issues)



The screenshot shows a web interface for Autokey Cipher encryption. It has two tabs: 'Embed' (active) and 'Extract'. Under 'Embed a message', there is a section 'Select a PNG image' with a placeholder 'Click to choose a PNG' and a note 'Success! Image downloaded.' Below this is a preview of a 'SONS OF ANARCHY' image. To the right, 'Your secret message' is 'HIDDEN', with 'Length: 6 | Capacity: 984150'. The 'Encryption type' is set to 'Autokey' (selected with a radio button). The 'Keyword (letters only)' is 'Message'. A blue button 'Embed & Download' is at the bottom.

Figure 3.3: Autokey Cipher encryption

3.3 Results Overall Discussion

We analyzed the performance of the **enCrypt** system across all tests. The Caesar and Vigenère ciphers consistently encrypted and decrypted messages correctly. The Autokey cipher worked in most cases but occasionally failed with white spaces. Steganography successfully concealed encrypted messages within PNG images without noticeable changes.

Overall, combining cryptography with steganography improved message security and confidentiality. The system proved reliable for secure communication in a web-based environment, though it has limitations with mobile browsers, lossy image formats, and certain Autokey cases. These results confirm that layered security increases protection while maintaining usability.

3.3.1 Complex Engineering Problem Discussion

The **enCrypt** project tackles the challenge of securing messages while hiding them in images. We combined cryptography and steganography to protect data and conceal its existence.

We faced issues with the Autokey cipher handling white spaces and ensuring hidden messages do not distort images. Hosting on GitHub also limited mobile browser access.

Despite these challenges, we built a system that encrypts, hides, and retrieves messages reliably, demonstrating how layered security addresses real-world communication problems.

Chapter 4

Conclusion

This project successfully demonstrates the integration of cryptography and steganography within a lightweight, browser-based tool. By implementing classical encryption techniques such as the Caesar, Vigenère, and Autokey ciphers in combination with Least Significant Bit (LSB) image steganography, the system achieves a **dual-layer protection mechanism**. This ensures that a message is not only encrypted but also hidden, providing both secrecy and confidentiality.

Although the chosen cryptographic algorithms are considered weak by modern security standards, their inclusion serves an important educational purpose. They effectively illustrate how fundamental encryption concepts can be paired with steganographic methods to build secure communication systems. From an academic perspective, the project highlights the practicality of applying classical theories in a modern web environment.

The simplicity of the system design, coupled with its compatibility across modern browsers, demonstrates the accessibility of such technologies without requiring heavy computational resources or server-side infrastructure.

4.1 Discussion

- The project successfully demonstrates how classical cryptographic ciphers (Caesar, Vigenère, and Autokey) can be integrated with image-based steganography to create a two-layer security mechanism for secret communication. This dual approach ensures that even if the hidden data is discovered, it remains unreadable without the proper decryption key.
- The implementation is entirely client-side, lightweight, and compatible with all modern browsers. Since it is built using HTML, Tailwind CSS, and JavaScript, the system does not require any external libraries or server-side processing, making it portable and easy to deploy.
- A key limitation is that the system currently supports only PNG images, as the method relies on lossless compression to preserve hidden data. Formats like JPEG introduce compression artifacts that can damage embedded information.

4.2 Limitations

While the **enCrypt** system demonstrates effective integration of cryptography and steganography, it has some limitations.

- **Autokey Cipher Issues:** The Autokey cipher sometimes fails to process messages with white spaces correctly. This can lead to errors in encryption or decryption for messages that include spaces.
- **Limited Image Format Support:** The system currently supports only PNG images. It cannot reliably handle lossy formats like JPEG, which can corrupt hidden data during compression.
- **Classical Cipher Limitations:** We use classical ciphers (Caesar, Vigenère, Autokey) for encryption, which are not secure against modern cryptanalysis. This limits the system's security in real-world applications.
- **Capacity Constraints:** The system can hide only a limited amount of data in images. Larger messages may require larger images or multiple files.
- **Platform Dependency:** The current web-based implementation may not be convenient for mobile users or offline usage, limiting accessibility.

4.3 Scope of Future Work

While the current version of the **enCrypt** system achieves its primary goals by integrating classical cryptography with image-based steganography, we can further improve and expand it. These enhancements will increase the system's robustness and broaden its applicability in real-world scenarios.

- **Support for Additional Image Formats:** Currently, the system only supports PNG images due to their lossless nature. We plan to extend compatibility to formats like JPEG, BMP, and GIF to improve usability. We will handle lossy formats such as JPEG carefully, as compression artifacts can affect hidden data.
- **Integration of Modern Encryption Algorithms:** We currently use classical ciphers (Caesar, Vigenère, Autokey) which serve educational purposes but are not secure against modern attacks. We will incorporate advanced encryption standards such as **AES (Advanced Encryption Standard)** and **RSA (Rivest, Shamir, Adleman)** to strengthen the system and approach real-world security levels.
- **Expanding to Multimedia Steganography:** We will extend our approach beyond images to audio and video files. We can embed secret messages within sound samples for audio steganography, and use multiple frames to hide data in video steganography. These methods will increase storage capacity and diversify the system's applications.

- **Mobile and Cross-Platform Applications:** We aim to develop dedicated mobile applications (Android/iOS) or cross-platform desktop clients. This will allow users to embed and extract messages securely on-the-go, removing reliance on a web browser.
- **Cloud Integration and Collaboration:** We will explore cloud-based storage and sharing features to allow secure exchange of encrypted stego-images online. We will also encourage community contributions through open-source collaboration to accelerate feature development and improvements.

Chapter 5

References

stylesuxx, *Steganography.js – A JavaScript Library for Hiding Information in Images*, Available at: <https://stylesuxx.github.io/steganography/>, Accessed: 2025-08-24.

Futureboy, *Steganography Online – Encrypt and Hide Text in Images*, Available at: <https://futureboy.us/stegano/encinput.html>, Accessed: 2025-08-24.

Faisal Hasan, Rokshanara Toma, Rafi Siddique, *enCrypt– Mr Robot inspired Message Encoder using Cryptography and Steganography*, Available at: <https://github.com/Faisalhasan7/enCrypt>, Accessed: 2025-08-24.