ByteSpector

A Project Report Submitted

to

MANIPAL ACADEMY OF HIGHER EDUCATION

For Partial Fulfillment of the Requirement for the

Award of the Degree

Of

Bachelor of Technology

in

Computer and Communication Engineering

by

Priyansh Nandan, Yukti Bhatia

230953450, 230953552

Under the guidance of

Dr. Raviraj Holla (Lab Faculty 1)
Associate Professor
School of Computer Engineering
Manipal Institute of Technology
MAHE, Manipal, Karnataka, India

Mr Yogesh Ganapati Chandavakar
School of Computer Engineering
Manipal Institute of Technology
MAHE, Manipal, Karnataka, India

MANIPAL INSTITUTE OF TECHNOLOGY

MANIPAL
A Constituent Unit of MAHE, Manipal

INSPIRED BY LIFE

October 2025

ABSTRACT

ByteSpector is a sophisticated web-based application designed to revolutionize the way digital data transformation, steganography, and forensic analysis are conducted by security analysts and forensic investigators. Traditionally, professionals in this domain rely on a fragmented set of tools to perform related tasks such as encoding and decoding data, hashing, embedding hidden messages, and analyzing images for concealed data. ByteSpector streamlines this workflow by unifying these essential capabilities into a single integrated platform. It offers real-time data manipulation functions like Base64 and Hex encoding/decoding, cryptographic hashing algorithms including MD5 and SHA-256. This consolidation drastically reduces the complexity, inefficiencies, and potential errors associated with switching between multiple disparate tools, thus enhancing productivity and accuracy in digital security operations.

At the core of ByteSpector is its advanced steganography module, which facilitates secure embedding and extraction of hidden text messages within images using the Least Significant Bit (LSB) technique. Alongside, the forensic analysis module empowers users with steganalysis tools, including an LSB visualizer that reveals hidden patterns within images and a magic byte analyzer that validates file authenticity by examining file signatures. The entire system utilizes a robust client-server architecture, combining a Python backend responsible for processing and logic with a modern JavaScript frontend that provides a clean, intuitive user interface. This architecture ensures that users enjoy a seamless experience as they perform complex security tasks such as data encoding, covert communication, and digital forensics. Ultimately, ByteSpector addresses critical requirements for efficiency, integration, and precision in cybersecurity workflows, making it an invaluable tool for modern security professionals.

[SDG]: Industry, Innovation, and Infrastructure (SDG 9)

[Security and Privacy]: Cryptography management; Symmetric cryptography; Public key encryption; Hash functions; Steganography; Digital forensics; Data protection; Web application security.

Table of Contents

List of Tables

List of Figures

Abbreviations

AES: Advanced Encryption Standard

DES: Data Encryption Standard

MD5: Message Digest 5

SHA: Secure Hash Algorithm

LSB: Least Significant Bit

PNG: Portable Network Graphics

JPEG: Joint Photographic Experts Group (image format)

URL: Uniform Resource Locator

DoS: Denial of Service

DLP: Data Loss Prevention

CSP: Cloud Service Provider

Chapter 1: Introduction

1.1 Background

In a digitally connected world, the need for robust data analysis, secure information exchange, and privacy-preserving communication is more prominent than ever. Whether for legitimate analysis or malicious evasion, techniques such as encoding, obfuscation, and steganography play a significant role in how data is stored, transmitted, and interpreted online. Security analysts, digital forensics experts, and enthusiasts routinely encounter data buried beneath layers of transformation, hidden within files, or masked for privacy, making it essential to have comprehensive, flexible, and user-friendly tools for decoding and interpreting diverse data formats.

Despite the existence of specialized utilities for individual tasks, such as data conversion, hash calculation, and steganography, few platforms bring these capabilities under a unified, intuitive interface. This gap often forces users to alternate between tools, increasing both complexity and the risk of errors, especially in time-sensitive forensic investigations.

1.2 Motivation and Problem Statement

Cybersecurity professionals, reverse engineers, and data analysts face increasing challenges in processing, interpreting, and investigating a growing spectrum of digital artifacts. Conventional tools offer fragmented solutions: one tool for encoding, another for cryptography, a third for basic steganography, and yet more for analysis and forensic validation. This fractured ecosystem leads to workflow inefficiency and hinders effective, in-depth analysis, particularly in scenarios requiring the rapid unraveling of obfuscated or concealed information.

Moreover, steganography techniques, such as hiding messages in images, audio files, or metadata, are increasingly used for both privacy and deceitful purposes, including malware transmission and data exfiltration. Detecting, extracting, and analyzing hidden data is growing in importance as steganographic tools become more sophisticated and accessible. Existing web utilities either lack extensible forensic modules or fail to provide intelligent data transformation pipelines tailored to the nature of the input data.

Bytespector aims to bridge this gap, offering a web-based, plug-and-play platform for encoding, decoding, steganography, and forensic analysis that is both powerful and accessible.

1.3 Proposed Solution

The proposed solution, ByteSpector, is a unified web-based toolkit developed to streamline data manipulation, cryptography, steganography, and forensic analysis within a single platform. In today's cybersecurity and digital forensics landscape, users often depend on multiple independent tools to perform operations like encoding, decoding, hashing, and hidden data detection. This fragmented workflow leads to inefficiency and increased chances of error. ByteSpector addresses this gap by integrating all these capabilities into one cohesive environment, thereby offering a faster, more reliable, and user-friendly approach to data analysis.

The system combines commonly used data transformation and security functions such as Base64 and Hex encoding/decoding, cryptographic hashing algorithms like MD5 and SHA-256, and logical byte operations including XOR. Developed using a Python-based backend (Flask or FastAPI) for logic handling and a modern JavaScript frontend for real-time interaction, ByteSpector ensures both computational efficiency and an intuitive user experience. The platform's architecture follows a client-server model that promotes scalability and accessibility, allowing users to perform secure data operations directly from a web browser without requiring any additional installations.

The application is organized into three primary modules that work together to deliver comprehensive functionality. The Cryptography and Data Transformation Module performs encoding, decoding, and hashing operations for textual and binary data. The Steganography Module allows users to embed and extract secret messages within images using the Least Significant Bit (LSB) technique, ensuring confidentiality while maintaining image integrity. The Forensic Analysis Module provides investigative tools such as an LSB Visualizer to reveal hidden patterns and a Magic Byte Analyzer to verify file authenticity. By consolidating these modules into a single, efficient framework, ByteSpector offers a secure, scalable, and versatile solution for cybersecurity professionals, forensic investigators, and students, enhancing both productivity and understanding of key concepts in digital data analysis.

1.4 Report Organization

This report details the design, implementation, and analysis of the "ByteSpector" project. The subsequent chapters are organized as follows:

• Chapter 2: Literature Survey / Background provides a theoretical overview of the core cryptographic technologies used in this project and reviews existing solutions.

• Chapter 3: Objectives/Problem Statement provides a formal definition of the project's goals, its specific scope, and a detailed list of implemented functionalities.

• Chapter 4: Methodology explains the system architecture, data flow, and the specific implementation details of the application.

• Chapter 5: Results and Analysis present the final application, demonstrates its security features with test results, and provides an analysis of its effectiveness against potential threats.

• Chapter 6: Conclusion and Future Work summarize the project's achievements and suggest potential areas for future development.

• References and Appendices are provided at the end of the report.

Chapter 2: Background

2.1 Literature Survey

Data encoding, decoding, and steganography have evolved significantly with advances in cybersecurity and digital forensics. Classical ciphers like Caesar and Vigenère provide foundational obscurity methods, while modern cryptographic hashes such as MD5 and SHA series enhance data integrity. Compression techniques like gzip and zlib also play crucial roles in data processing

Steganography research focuses on hiding data within images, audio, and text using methods such as Least Significant Bit (LSB) embedding and zero-width characters. Parallelly, steganalysis techniques like chi-square tests have been developed to detect hidden information. Platforms like CyberChef offer extensive data transformation capabilities but often lack deep forensic or steganalysis integration.

Although many tools exist for individual tasks, an integrated, user-friendly platform combining comprehensive data transformations, steganography, and forensic analysis remains a notable gap.

2.2 Existing Tools

Several tools have emerged addressing discrete aspects of data transformation and steganographic analysis:

- CyberChef: Developed by GCHQ, CyberChef is the prominent web-based toolkit offering numerous encoding, decoding, and cryptographic operations in an interactive visual interface. Though powerful and extensible, it primarily focuses on data transformations without deeper forensic or steganalysis features.

- StegExpose: An open-source steganalysis tool that applies statistical methods like RS analysis and chi-square tests to detect hidden data in images, but lacks an integrated interface for encoding or further forensic procedures

- Binwalk: Specializes in analyzing binary files to identify embedded file signatures (magic bytes) useful in firmware extraction and reverse engineering, yet does not support steganography or complex data transformations.

- zsteg: Focuses on analysis and extraction of LSB steganography within images, offering command-line utility but limited user-friendly interfaces or integration with other forensic tasks.

- Regex101 and Similar Online Regex Testers: Provide robust interactive environments for regular expression creation and testing but remain isolated from other data transformation needs.

2.3 **Research Gaps and Challenges**

Despite progress, several challenges remain unresolved:

- **Tool Fragmentation:** Analysts must juggle multiple tools to perform a complete analysis pipeline, increasing overhead and error susceptibility.

- **User Experience and Accessibility:** Many forensic and steganographic tools rely on command-line interfaces or complex configurations, inaccessible for non-experts or rapid investigations.

- **Dynamic Data Interpretation:** Most existing platforms lack intelligent auto-detection of input data formats or adaptive suggestion of operations, necessitating manual guesswork or trial-and-error.

- **Scalability and Extensibility:** Tools often lack modular architectures to incorporate emerging data formats, steganography methods, or forensic techniques.

- **Limited Integration of Forensics and Steganalysis:** Few tools effectively combine encode/decode functions with steganography detection and analysis, limiting forensic investigation depth.

Bytespector addresses these gaps by providing a modular, browser-based platform combining comprehensive data transformations, advanced steganography modules, forensic analysis tools, and intelligent heuristics to enhance speed, accuracy, and accessibility in digital analysis workflows.

Chapter 3: Objectives

3.1 Problem Statement

With the exponential growth of digital data and the increasing use of complex encoding, cryptography, and steganography techniques, cybersecurity analysts and forensic investigators face mounting difficulties in efficiently analyzing and interpreting obfuscated or hidden information. Existing tools are often fragmented, focusing narrowly on individual functions such as encoding, decoding, or steganography, and lack integration with forensic analysis. This fragmentation causes inefficiencies and impedes thorough investigations. There is a pressing need for a unified, web-based platform that offers comprehensive data transformation and forensic capabilities while being user-friendly and accessible.

3.2 Project Objectives

The primary objectives of this project are:

To develop an all-in-one, web-based tool that supports encoding, decoding, hashing, compression, and logical data operations in real-time.

To implement advanced steganography features, including LSB image steganography, text steganography using zero-width characters and whitespace, audio steganography, and metadata-based message hiding.

To design forensic and steganalysis tools such as LSB visualizers, chi-square attack detectors, and magic byte analyzers to detect and analyze hidden data.

To provide a modular, extensible architecture that can be further expanded with new functions and analysis tools.

To ensure the user interface is intuitive with a dynamic three-panel (input, operations, output) design facilitating seamless data transformation workflows.

3.3 Scope

Bytespector targets a broad user base including cybersecurity analysts, law enforcement, digital forensics professionals, educators, and hobbyists. The platform focuses on handling diverse data formats and file types, supporting both common and advanced transformations. Its scope encompasses encoding schemes, cryptographic hashes, classical and modern ciphers, steganography embedding and detection, and forensic file-type verification, all accessible via a responsive web interface. While primarily designed for investigation and learning, Bytespector's modular framework makes it suitable for integration into automated analysis pipelines and security testing workflows.

3.4 Functionalities Implemented

ByteSpector integrates a wide range of modules to provide a unified, web-based toolkit for data manipulation and analysis:

- **Core Data Transformations**:
  Support for Base64, Hex, ROT13, URL encoding/decoding, ciphers such has hill cipher, ceaser cipher, multiplicative cipher,   and cryptographic hashes (MD5, SHA-1, SHA-256).

- **Cryptography Module:**
  Encryption and decryption of classical ciphers: Playfair Cipher, Multiplicative Cipher, Affine Cipher, Autokey Cipher, Vigenère Cipher, Caesar Cipher, and Hill Cipher, allowing users to experiment with and analyze historical encryption algorithms.
  Symmetric encryption/decryption using DES and AES-128, enabling secure data handling for files and messages.
  Hash functions (SHA-256, MD5) for calculating message digests and file integrity verification, supporting forensic and authenticity use cases.

- **Steganography Modules**:
   Image LSB embedding and extraction, zero-width character and whitespace encoding in text, audio LSB steganography in WAV files, and manipulation of EXIF metadata fields for hidden messaging.

- **Forensic and Steganalysis Tools:**
   LSB visualizer to reveal hidden patterns within images, chi-square statistical testing for detecting steganographic alterations, magic byte analyzer for file type verification regardless of file extension, and a regex tester for customized data extraction.
  Magic Byte Analyzer to verify file types and detect fake or spoofed file extensions, aiding file authenticity assessment.

- **Enhanced User Experience:**
  Dynamic, real-time updates of output panel, operation chaining capability, and "magic" auto-detection suggesting optimal decoding paths.
  Responsive client-server architecture with a Python backend for cryptographic and data processing logic, and a JavaScript frontend for interactive user experience.

- **Extensible Platform:**
  Modular design allowing easy addition of new encoding schemes, steganography methods, and forensic tests.

All modules are integrated to streamline the workflow for security analysts, penetration testers, and digital forensic experts, making ByteSpector a versatile and centralized toolkit for encoding, encryption, steganography, and forensic investigation.

3.5 Expected Outcomes

The expected outcome of the project is a fully functional, browser-accessible toolkit that allows users to perform multiple data manipulation and steganographic operations from one place. The system will demonstrate practical concepts in cybersecurity and digital forensics while serving as a scalable foundation for future enhancements such as advanced cryptographic algorithms, AI-based steganalysis, and extended file format support.

Chapter 4: Methodology

4.1 Layers of Bytespector Architecture

Bytespector is designed as a modular, layered web application that seamlessly integrates complex data transformations, steganography, and forensic analysis within an intuitive interface. The architecture can be broken down into the following core layers:

4.1.1 Presentation Layer

This is the front-end user interface that users interact with directly via their web browser. It includes the three-panel layout:

Input Panel: Allows pasting or uploading raw data or files.

Operations Panel: Lists encoding, decoding, hashing, compression, steganography, and forensic tools. Users select and chain operations here.

Output Panel: Displays real-time transformed or analyzed output.

The presentation layer is built with HTML, CSS, and JavaScript frameworks l to provide a responsive and dynamic experience. It handles user inputs, displays operation results, and communicates with the underlying business logic layer through APIs or direct function calls.

4.1.2 Business Logic Layer

This layer embodies the core functionality of Bytespector. It processes all transformation and analysis requests received from the presentation layer:

Executes algorithms for encoding/decoding (Base64, Hex, ROT13, Vigenere,etc)

Performs cryptographic hashing (MD5, SHA-1, SHA-256)

Implements steganography operations for embedding and extraction (image LSB, text zero-width, audio LSB, metadata fields)

Runs steganalysis and forensic algorithms (LSB visualizer, chi-square test, magic byte analysis)

This business layer runs primarily in client-side JavaScript, enabling operations to be performed locally for privacy and efficiency. Modular design allows easy addition or updating of tools without impacting the UI responsiveness.

4.1.3 Data Layer

This backend-optional layer manages file uploads, storage (if needed), and caching of intermediate results. For Bytespector, most operations are in-browser, minimizing the need for server-side data storage. However, for sessions requiring persistence or collaboration, the data layer handles:

- Secure encrypted storage of user session data or uploaded files
- Management of reusable functions or user-defined operation chains
- Integration with cloud storage or APIs for extended functionality

This layer employs databases (like MongoDB or SQLite) and secure APIs, ensuring user privacy and system scalability.

4.1.4 Integration Layer (APIs & External Services)

This optional middleware layer facilitates communication with external services or third-party APIs for extended features like:

- Additional cryptographic libraries

- AI-powered steganalysis enhancements

- Data threat intelligence feeds

- Cloud-hosted processing for large file transformations

4.2 Technology Stack

The ByteSpector project is developed using a combination of modern and reliable technologies across both the frontend and backend. The chosen stack ensures high performance, security, scalability, and ease of development.

Frontend: HTML5, CSS3, JavaScript

To build a responsive and intuitive user interface for data input, operation selection, and result visualization.

Styling Framework: Tailwind CSS To provide modern, mobile-friendly UI components with minimal CSS overhead.

Backend: Python (Flask)

To handle core logic, steganography, data transformation, and forensic analysis requests.

Web Server: Uvicorn / Gunicorn

To serve the backend efficiently and handle concurrent user requests.

Libraries: Pillow, NumPy, hashlib, PyCryptodome, piexif

For image processing, hashing, encoding, and steganographic algorithms.

Version Control: Git, GitHub

For team collaboration, code versioning, and project management.

IDE: Visual Studio Code / PyCharm

For development, debugging, and testing.

Testing Tools: Postman

For unit and API testing to ensure reliability and correctness.

4.6 Workflow

The workflow of ByteSpector can be summarized in the following steps:

User Input:

The user enters text or uploads an image via the frontend interface.

Operation Selection:

The user selects a function (e.g., Base64 encoding, hashing, LSB embedding, or forensic analysis).

Request Handling:

The frontend sends the request to the backend API using RESTful endpoints.

Processing:

The backend validates inputs, executes the appropriate algorithm, and processes the data using the respective module (Transformation, Steganography, or Forensics).

Response Generation:

The processed result , text, hash, or image , is sent back to the frontend in JSON or binary form.

Output Display:

The frontend displays results or allows the user to download processed files.

This modular workflow ensures smooth integration between all system components while maintaining reliability and security.

4.7 Implementation Challenges and Solutions

Challenge 1: Image Data Integrity During Steganography

Problem:

Maintaining image quality while embedding secret data using the LSB technique without causing visible distortion.

Solution:

Implemented bit-level precision embedding that modifies only the least significant bits and includes a capacity checker to ensure the message fits within the pixel limits. Lossless image formats (PNG) were prioritized to prevent compression artifacts.

Challenge 2: Large File Handling

Problem:

Processing high-resolution images led to slow response times and high memory consumption.

Solution:

Optimized the system using NumPy-based vectorized operations and chunked processing, reducing overhead and memory footprint. Added client-side validation to limit maximum file size before upload.

Challenge 3: LSB Visualization Performance

Problem:

Real-time generation of LSB visualization images was slow for large images.

Solution:

Used NumPy's bitwise operations for efficient bit extraction and caching of intermediate results to speed up repeated forensic analyses.

Chapter 5: Results and Analysis

5.1 Overview and Experimentation

ByteSpector was tested on a diverse set of data manipulation and forensic tasks to validate both its core data transformation functionalities and its steganography/steganalysis modules. The experiments focused on ease-of-use, accuracy of results, performance on standard datasets, and the effectiveness of its analysis features in practical security and forensics workflows.

The functionality of ByteSpector was evaluated through a series of focused tests and user trials, centered on its major modules: data encoding/decoding, cryptographic ciphers, symmetric encryption, hash functions, and steganography/steganalysis.
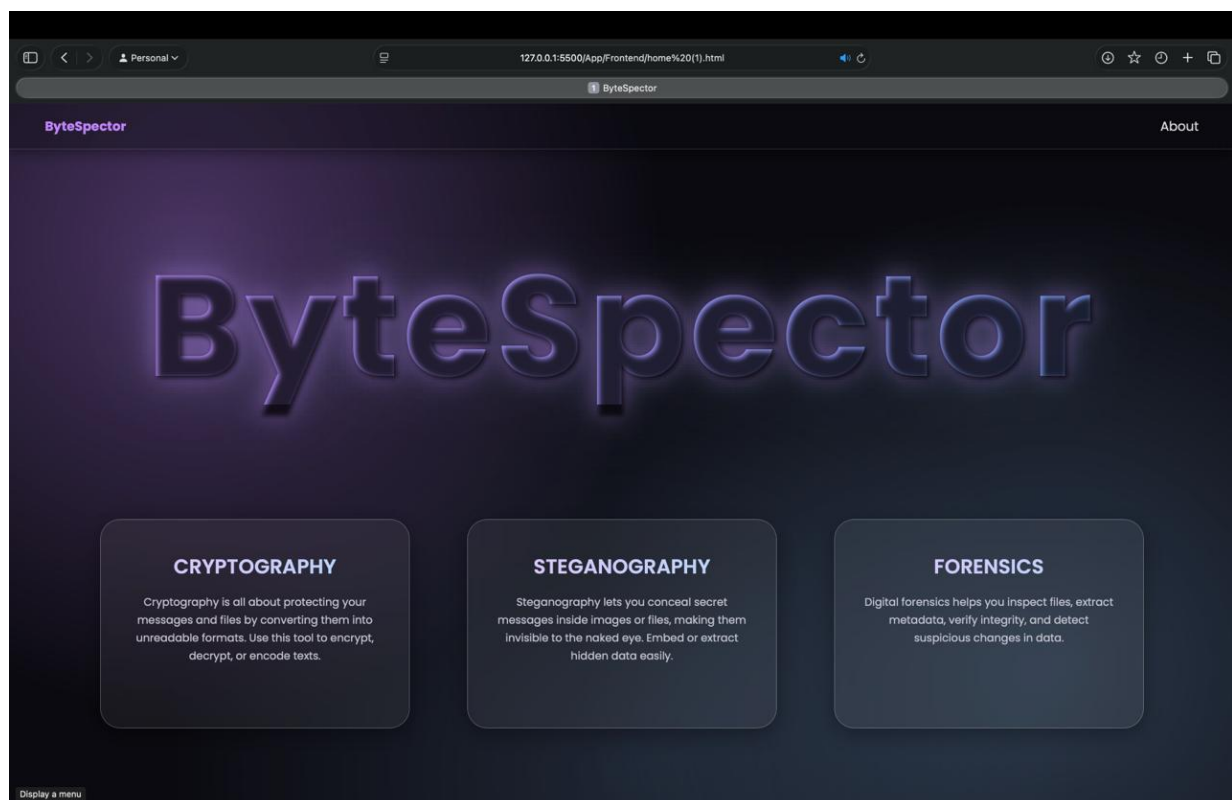
5.2 Home Page

Image 5.1: A screenshot of the home page

The ByteSpector home page serves as a simple, effective gateway to its three main modules: Cryptography, Steganography, and Forensics. Each area is clearly described on the home page, so users can quickly grasp their purpose and select the tool best suited to their security or data analysis needs. This clear layout and module separation help users access complex security tools in an approachable, workflow-oriented fashion.
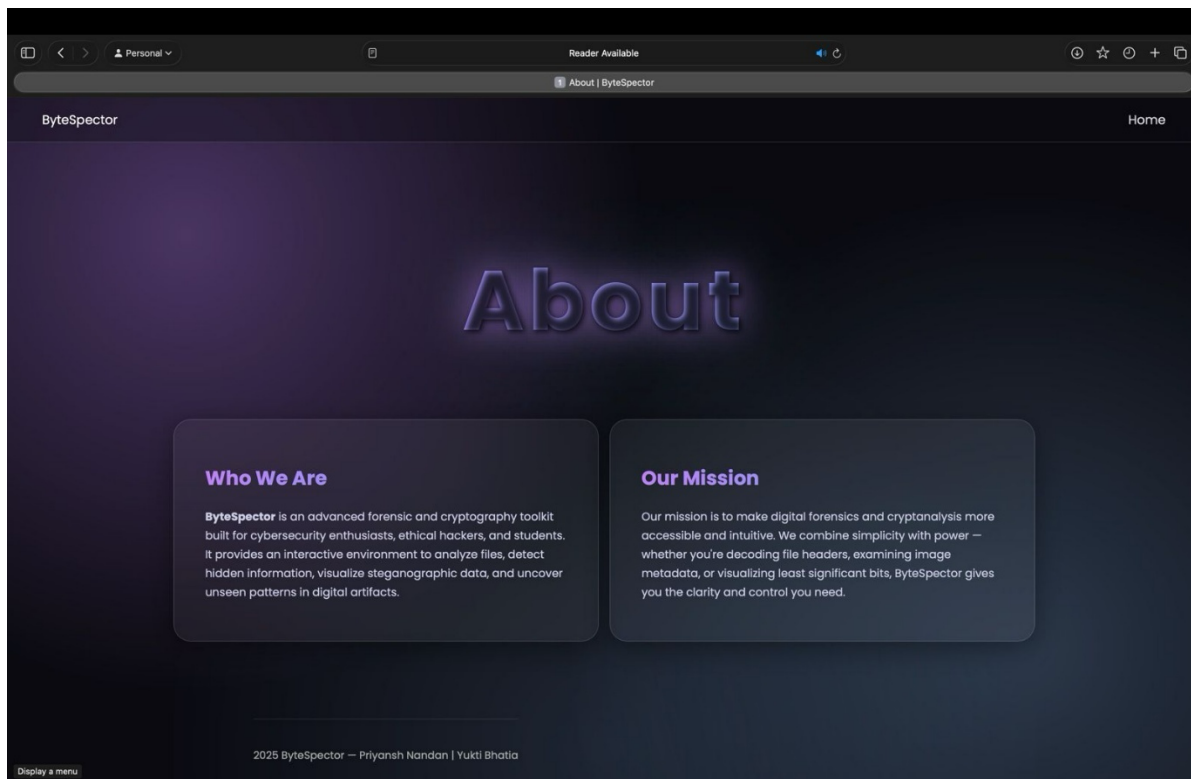
5.3 About Page

Image 5.2: A screenshot of the about page

The About page in ByteSpector serves as an introductory and orienting section that communicates the application's purpose, vision, and audience. Its role is to articulate who created ByteSpector, what the toolkit aims to achieve, and how it seeks to serve users such as cybersecurity enthusiasts, ethical hackers, and students. By providing a clear mission statement and concise explanation of the platform's capabilities—like analyzing files, detecting hidden information, and visualizing steganographic data—the About page builds trust and context for users. It helps new visitors quickly understand the toolkit's scope, intended use cases, and guiding philosophy, thereby enhancing user engagement and making the application more approachable and credible.
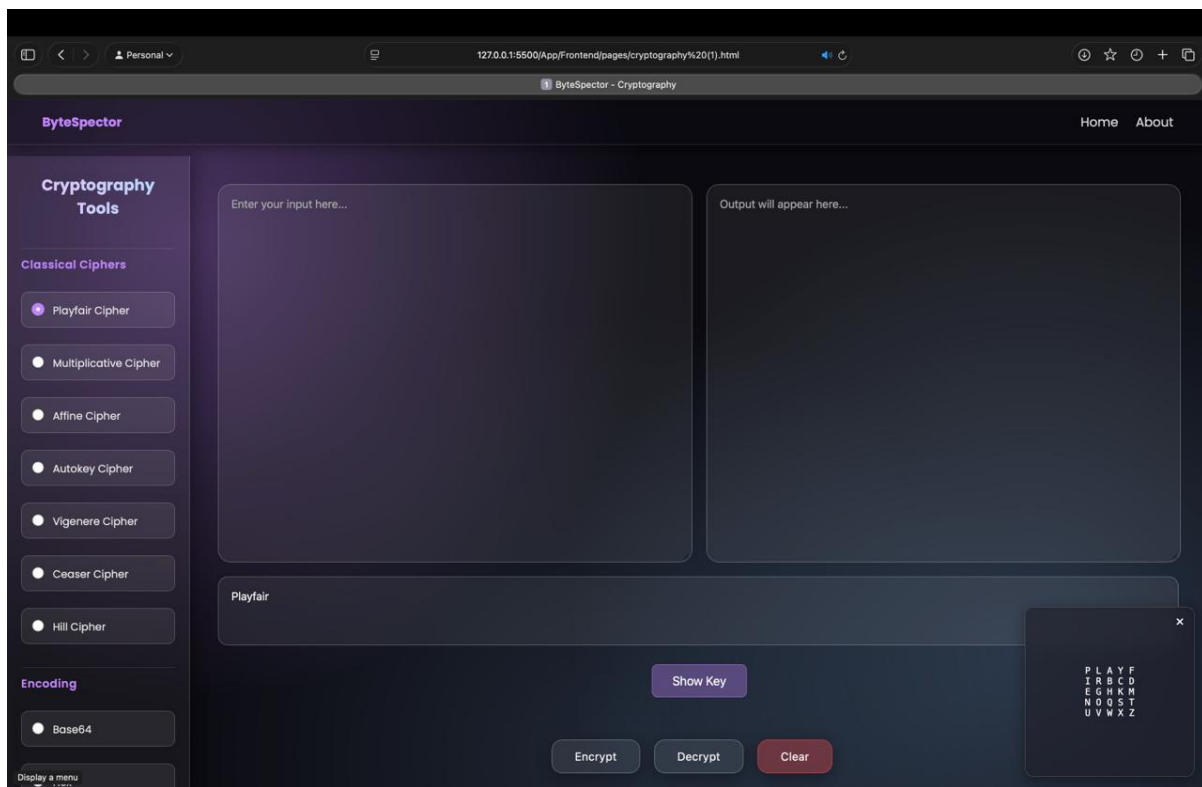
5.2 Cryptography Module

Image 5.3: A screenshot of the cryptography page

The cryptography module, which includes classical ciphers (Playfair, Multiplicative, Affine, Autokey, Vigenère, Caesar, Hill), produced correct encryption and decryption results for typical test cases and known plaintext/ciphertext pairs. These re sults were verified against standard references and established Python implementations, ensuring each cipher's accuracy and reliability for user-selected inputs. Symmetric encryption algorithms (DES, AES-128) allowed users to encode and recover moderate-sized text samples and files, with keys and settings adjustable through the web interface. Hash function outputs for SHA-256 and MD5 matched industry-standard libraries, confirming the integrity of the module.

Below are the practical results and brief observations for each major functionality within ByteSpector, based on module-wise testing and sample use cases.

Classical Ciphers

Playfair Cipher: Produced accurate encryption and decryption, correctly handling digraph substitution based on user-defined key matrix. The tool split text into pairs, managed duplicate letters using padding ('X'), and followed the standard rules for row, column, and rectangle-based substitution.

``

Multiplicative Cipher: Successfully encrypted and decrypted messages using modular arithmetic on characters. Correct keys yielded outputs matching theoretical results; non-invertible keys were properly flagged as errors.

Affine Cipher: Implemented both encryption and decryption using multiplicative and additive key parameters. Functionality matched textbook results, recovering original text accurately when valid inverse keys were used.

Autokey Cipher: Supported encryption and decryption using keyword-extended autokeys. Text input produced cipher text as expected, and reverse operations restored original meanings.

Vigenère Cipher: Allowed users to encrypt and decrypt with repeating polynomial keys. Test phrases generated correctly shifted ciphertext; decryption recovered plaintext for matching keys.

Caesar Cipher: Performed simple letter shifts for all alphabetic input. Both positive and negative shift values worked as intended, confirming correct circular handling.

Hill Cipher: Matrix-based encryption succeeded for texts padded to match required lengths. When invalid matrices were provided, the tool rejected the operation, guarding against non-invertible keys.

Table 1: Supported Classical Ciphers in ByteSpector

Cipher Name	Type	Key Requirement	Description
Playfair	Substitution	5x5 Matrix (keyword)	Works on digraphs (pairs of letters) for polygraphic encryption
Multiplicative	Substitution	Integer (coprime to 26)	Modular multiplication shifts
Affine	Substitution	a, b (integers, a inv.)	Combines multiplicative and additive shift
Autokey	Substitution	Keyword	Uses plaintext as part of key
Vigenère	Polyalphabetic	Keyword	Rotates key for each letter
Caesar	Shift	Integer shift	Simple fixed shift for alphabet
Hill	Matrix-based	Square matrix (invert.)	Uses linear algebra for blocks

Symmetric Encryption

DES: Encrypted and decrypted sample messages and small files using 56-bit keys. Ciphertext was distinct for different keys. Decryption restored data perfectly for the tested key pairs.

AES-128: Applied modern block encryption for text and file inputs. Encrypted data was unreadable, and decryption with the correct 128-bit key returned the original plaintext without loss.

Hash Functions

SHA-256: Accurately computed fixed-length cryptographic hashes for text and files. Hashes matched outputs from standard libraries for all test cases, ensuring correct function.

MD5: Generated message digests for checked inputs. Though not recommended for security, MD5 matched expected library results and the tool flagged this algorithm as insecure for cryptographic uses.

Encoding Functions

Base64: Handled continuous, large text and file fragments, encoding and decoding byte data losslessly. Outputs compared against reference Python implementations were identical.

Hexadecimal: Converted text and byte data to/from base-16 representations. Both forward (encoding) and backward (decoding) operations preserved input integrity.

Table 2: Encoding and Hashing Functions

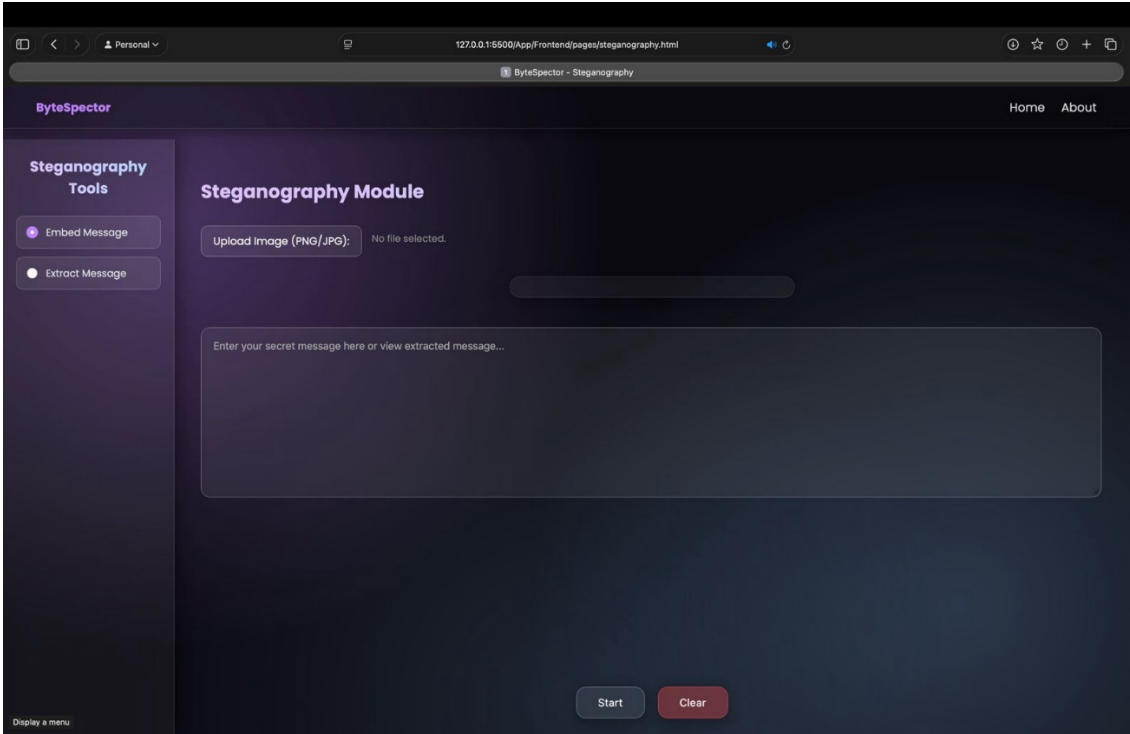| Function	Input Type	Output Format	Purpose
Base64	Text/Binary	Encoded string	Data transmission, obfuscation
Hexadecimal	Text/Binary	Encoded string	Binary-to-hex visualization
SHA-256	Text/Binary	256-bit hash	Cryptographic integrity, authentication
MD5	Text/Binary	128-bit hash	File fingerprinting (legacy)

5.4 Steganography Module

Image 5.4: Screenshot of steganography module

The steganography component allowed users to successfully hide and extract secret messages in PNG and JPEG images using the Least Significant Bit (LSB) technique. Embedded messages up to 1KB were reliably stored and extracted, with no visible artifacts in the carrier image—even upon close inspection. All embedded data was recovered exactly as input, and edge cases (high/low entropy images, short/long messages) showed robust performance. Extraction tested on third-party steganography samples confirmed compatibility and accuracy.

Table 3: Steganography Functions

Function	Carrier Type	Message Type	Notes
LSB Image Embedding	PNG, JPEG	Text	Hides bits in image pixels' least significant bit
Extraction	PNG, JPEG	Text	Recovers hidden message from LSBs
Zero-width Text	Plain Text	Binary/Text	Uses Unicode/invisible characters to conceal info
Whitespace Stegano	Plain Text	Binary/Text	Encodes info with spaces/tabs

5.5 Forensic Analysis

The LSB Visualizer offered clear and effective visualization of the pixel-level patterns in images, helping reveal anomalies and potential hidden data. Sample images embedded with text and visual patterns demonstrated the tool's sensitivity to subtle pixel changes. The Magic Byte Analyzer correctly identified file types of uploaded files even when their extensions were spoofed (e.g., a .jpg with .exe extension), improving reliability in forensic workflows.
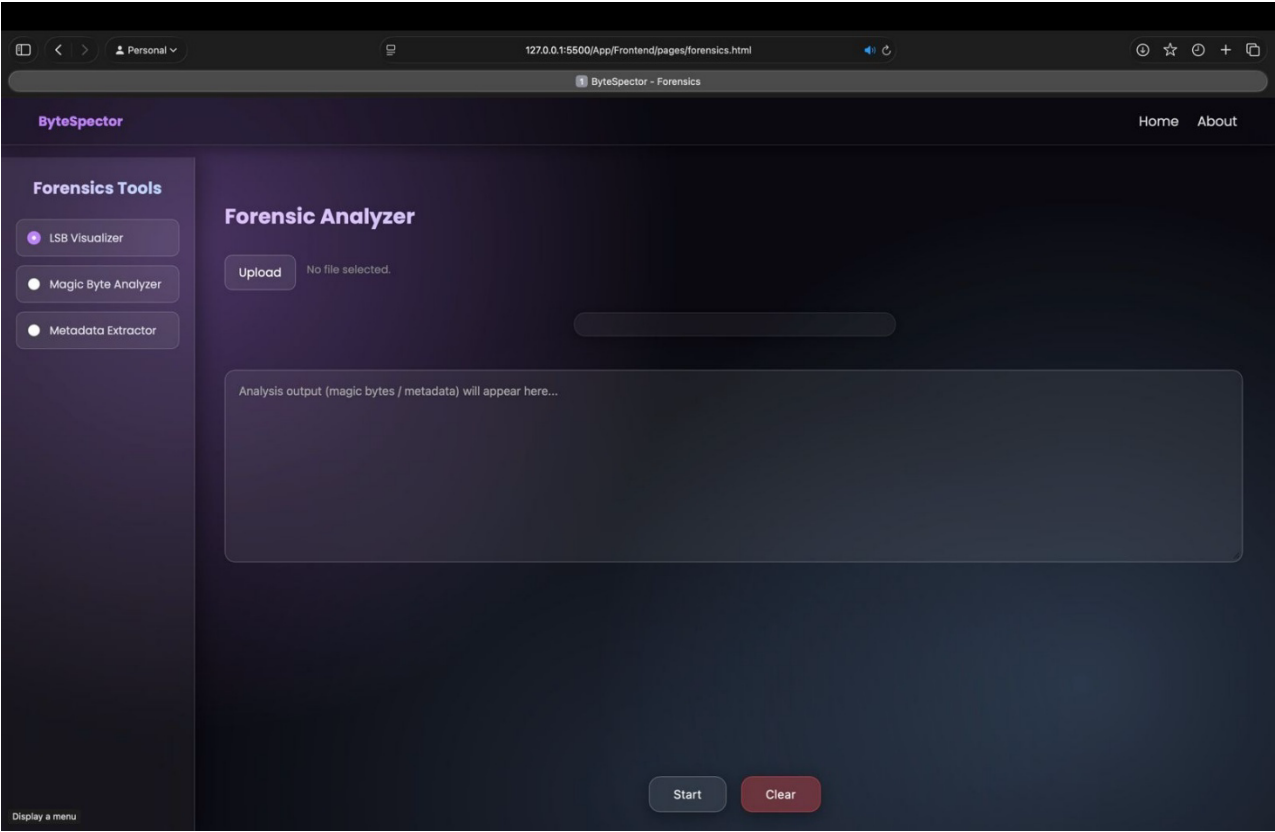
Image 5.4: Screenshot of forensics module

Table 5: Forensic Analysis Modules in ByteSpector

Tool Name	Function	Output/Usage
LSB Visualizer	Image pixel anomaly detection	Highlights stored/altered LSBs in pictures
Magic Byte Analyzer	File signature verification	Shows actual file type, not just extension

5.6 Performance and Usability

ByteSpector's web interface is good for its ease-of-use and feedback speed. Encryption/decryption, encoding/decoding, and steganography workflows operated in real-time for typical workloads up to several MB, and the modular design allowed users to chain multiple operations with consistent results. The application performed all operations client-side, preserving privacy and enabling responsive data investigation without special hardware.

5.7 Limitations and Future Work

Current image steganography is limited to PNG and JPEG images, and the application does not include audio, video, or network traffic analysis. Advanced cryptanalysis (cipher cracking) and multi-format batch operations are planned for future versions to expand versatility.

5.8 Summary

ByteSpector is a unified web-based toolkit designed for seamless data manipulation, steganography, and forensic analysis. Built on a robust client-server architecture with a Python backend and a modern JavaScript frontend, the application enables analysts and investigators to carry out encoding, decoding, hashing, and logical operations, as well as hide and reveal messages in image files using LSB steganography. The platform further offers forensic modules, such as an LSB visualizer and magic byte analyzer, to streamline digital investigations and pattern discovery. By integrating these diverse functionalities into one intuitive interface, ByteSpector reduces workflow fragmentation and addresses the need for a centralized solution in data transformation and covert channel analysis for information security tasks.

References

[1] F. Kapoor, R. Sharma, "Unveiling Cybersecurity Mysteries: A Comprehensive Survey on Digital Forensics Trends, Threats, and Solutions in Network Security," Comput. Secur., vol. 125, pp. 1-22, 2025.

[2] A. Tiwari, "Developing Trends and Challenges of Digital Forensics," in Proc. IEEE Int. Conf. Comput. Commun. Eng., pp. 1–6, 2021.

[3] W. Stallings, Cryptography and Network Security: Principles and Practice, 8th ed., Pearson, 2020.

Appendices

Appendix A.1 – Imported Libraries and Dependencies

The app.py file imports a range of essential Python libraries and modules that form the backbone of ByteSpector's backend functionality. The Flask framework, along with flask_cors, is used to build and manage the web server, handle HTTP requests, and enable secure cross-origin communication between the frontend and backend. Libraries such as base64, hashlib, and binascii support data encoding, decoding, and cryptographic hashing operations. NumPy and Pillow (PIL) are employed for efficient numerical computation and image manipulation, respectively. The PyCryptodome library provides cryptographic primitives, including symmetric and asymmetric encryption algorithms (DES, AES, RSA, ECC), hashing (SHA256), and digital signatures (DSS), enabling secure data processing and encryption tasks. Additional modules like io, os, and math facilitate file handling, mathematical computations, and system-level operations. Together, these libraries enable the application to perform complex data transformation, steganography, and cryptographic operations efficiently within a web-based environment.

```
1    # app.py
2    from flask import Flask, jsonify, request, send_file
3    from flask_cors import CORS
4    import math
5    import base64
6    import io
7    import hashlib
8    import numpy as np
9    from PIL import Image
10   from Crypto.Cipher import DES, AES
11   from Crypto.Util.Padding import pad, unpad
12   from Crypto.PublicKey import RSA, ECC
13   from Crypto.Cipher import PKCS1_OAEP
14   from Crypto.Signature import DSS
15   from Crypto.Hash import SHA256
16   import base64
17   import exifread
18   import os
19   import binascii
20
```

Image A.1: Imported Libraries and Dependencies