

DECLARATION

I Aliyu Abubakar Tukur, a student of the department of Computer Science with registration number 17/47719U/1, Abubakar Tafawa Balewa University Bauchi, hereby declare this project titled **“Data encryption with image and audio steganography: a comparative study”** has been carried out by me under the supervision of MR ISMAIL ALIYU and that it has never been submitted or presented for award of any degree in any form to any institution. All sources of material toward realizing this project are specifically acknowledged via the references provided.

CERTIFICATION

This project titled “**Data encryption with image and audio steganography: a comparative study**” written by Aliyu Abubakar Tukur a student of Computer science department with registration number 17/47719U/1 meets the requirements governing the award of the degree of Bachelor of Technology in Computer Science and is approved for its contribution to knowledge and literary representation.

Mr. Ismail Aliyu
(Project Supervisor)

Date

Dr. Fatima Umar Zambuk
(Head of Department)

Date

Dr. S.M Aliyu
(External Supervisor)

Date

ACKNOWLEDGEMENT

My appreciation also goes to the entire staff of Department of Computer Science for the required discipline I gained which actually shaped my ability towards this great achievement.

My profound gratitude goes to Almighty Allah who saw me through my Degree program and has enabled me to embark on this Project. He has been my all in all. I am grateful to my supervisor for checking on me to make sure that I was learning and doing the appropriate work needed to tackle the current world challenges as they come. My utmost gratitude goes to my entire family especially my older brother Engr. Adamu Abubakar Tukur who has been a backbone in my life and also a role model to me since childhood, following him is Geo. Sunusi Abubakar Tukur and Sir. Nasiru Abubakar Tukur who worked really hard to ensure that my life is a success, I appreciate you all. I am grateful to the Head of Department, Computer Science, Dr. Fatima Umar Zambuk for her advice and the encouragements she gave to students since the beginning of the journey. I am also grateful to my Departmental lecturers for being up and running to see that this degree program became a success. May God replenish your energy and bless you all.

DEDICATION

I dedicate this work to Almighty Allah who has been my source of inspiration, wisdom, guidance, knowledge and understanding. To Him I give all the glory, and also to my parents Alh. Abubakar Tukur and Hajiya Aishatu Abdulrahman and also my older brothers who have been there for me and helped me financially and morally.

ABSTRACT

The escalation in data breaches in recent years emphasize the need for stronger data protection mechanisms. This project introduces a hybrid model that integrates encryption with steganography to enhance the security and concealment of sensitive information. The system is tested on two mediums “image and audio” where encrypted data is embedded using steganography. A comparative analysis is conducted on the time taken for data embedding, file size before and after embedding, and the reliability of data extraction and decryption. The findings reveal that both image-based and audio-based steganography provide secure solutions for data transmission, but with notable tradeoffs in terms of speed, storage, and capacity. This hybrid approach offers an enhanced security model for protecting sensitive data in digital communications.

Table of content

DECLARATION	i
CERTIFICATION	ii
ACKNOWLEDGEMENT	iii
DEDICATION	iv
ABSTRACT	v
CHAPTER ONE	1
1.0 INTRODUCTION	1
1.1 Background of the Study	1
1.2 Statement of the Problem	1
1.3 Aim and Objectives	2
1.4 Scope and Limitation	2
Scope:	2
Limitations:	3
CHAPTER TWO	4
2.0 LITERATURE REVIEW	4
2.1 Introduction	4
2.2 Review of Related Work	5
2.3 Data Security and Encryption	6
2.4 Steganography	7
2.5 Hybrid Models	7
2.6 Image vs. Audio Steganography	7
2.7 Overview of the existing system	8
2.8 Problems Associated with the Existing System	8

CHAPTER THREE	11
3.0 METHODOLOGY	11
3.1 AES design	12
3.2 Steganography block diagram	12
3.3 AES block diagram	13
3.4 Phases	14
3.5 Tools and Technologies	17
3.6 AES Encryption	17
3.7 LSB Steganography	17
3.8 Performance Metrics	17
3.9 Use case diagram	19
CHAPTER FOUR	20
4.0 IMPLEMENTATION AND RESULT	20
4.1 IMPLEMENTATION	20
4.2 RESULT	27
CHAPTER FIVE	30
5.0 SUMMARY, CONCLUSION AND RECOMMENDATION	30
5.1 Summary	30
5.2 Conclusion	31
5.2 Recommendations	31
REFERENCES	33
APPENDIX	34

Table of figures

Figure 1: Methodology	
Figure 2: AES encryption class	20
Figure 3: image steganography class	21
Figure 4: Audio steganography class	22
Figure 5: GUI implementation	23
Figure 6: GUI implementation ii	24
Figure 7: GUI implementation iii	25
Figure 8: GUI implementation iv	26
Figure 9: GUI implementation v	26
Figure 10: hybridmodelcomparison main class	26
Figure 11: running program	27

CHAPTER ONE

1.0 INTRODUCTION

This project aims to develop and compare two hybrid models for data security by integrating encryption with steganography. The focus is on embedding encrypted data within image and audio files, evaluating the effectiveness, robustness, and practicality of each method. The comparison will be based on data capacity, perceptual impact, robustness to compression, and computational efficiency.

1.1 Background of the Study

With the increasing need for secure communication in today's digital world, it is essential to explore and implement advanced methods for protecting sensitive information. Cryptography and steganography are two key techniques used for data security. Cryptography ensures the confidentiality and integrity of data through encryption, while steganography hides the existence of the data itself. By combining these two techniques, we can create robust hybrid models that provide enhanced security.

This project explores the potential of combining Advanced Encryption Standard (AES) encryption with steganography to create a more robust security solution. By embedding encrypted data within cover files (images and audio), the hybrid model enhances both the confidentiality and the concealment of information during transmission.

1.2 Statement of the Problem

While encryption ensures the confidentiality of data, its presence can be easily detected, making it a target for interception and decryption attempts.

Existing data security methods either focus on encryption or steganography individually. There is a need to explore the combined use of these techniques to leverage their strengths and mitigate their weaknesses. This project seeks to develop and compare hybrid models that use encryption alongside image and audio steganography. Combining these two techniques provides a dual-layer defense. However, questions remain about which medium, image or audio offers better performance in terms of capacity, embedding speed, and data integrity.

1.3 Aim and Objectives

This project aims to develop and compare two hybrid models for data security by integrating encryption with steganography. This project aims to develop a hybrid model that integrates encryption and steganography for securing data and compares the performance of embedding data in images and audio files. The objectives are as follows:

- i. To encrypt sensitive data using AES.
- ii. To embed the encrypted data into image (.png) and audio (.wav) files using Least Significant Bit (LSB) steganography.
- iii. To evaluate the embedding time, file size changes, and accuracy of data retrieval for both mediums.
- iv. To identify the advantages and limitations of each hybrid model.

1.4 Scope and Limitation

Scope:

Data Types: This project will focus on hiding textual data within images(.png) and audio(.wav) files.

Steganographic Techniques: The primary techniques used will be Least Significant Bit (LSB) for both images and audio.

Encryption Algorithm: Advanced Encryption Standard (AES) will be used for encrypting data.

Evaluation Metrics: Embedding time, file size variations, and accuracy of data extraction for both mediums will be evaluated.

Implementation: The project will be implemented in Java, utilizing available libraries for image and audio processing.

Comparison: The project will compare the two hybrid models (image-based and audio-based) on predefined metrics.

Limitations:

Simplicity of Steganographic Techniques: The project will use basic LSB steganography, which might not be the most robust method against sophisticated steganalysis.

File Formats: The project will be limited to PNG for images and WAV for audio files to avoid complexities associated with different file formats.

Computational Resources: The project will be conducted using standard computational resources available in a typical undergraduate setting, which might limit the scale of testing.

Scope of Data: The project will focus on hiding relatively small amounts of data, suitable for proof of concept rather than large-scale implementation.

Other encryption and steganographic techniques are outside the scope of this investigation. Additionally, the impact of different image or audio qualities is not considered in the performance evaluation.

CHAPTER TWO

2.0 LITERATURE REVIEW

2.1 Introduction

Cryptography is the practice of securing information by converting it into a format that is unreadable to unauthorized users. The primary goal of cryptography is to protect data confidentiality, integrity, and authenticity. Two widely used cryptographic algorithms are the Advanced Encryption Standard (AES) and the Rivest-Shamir-Adleman (RSA) algorithm.

AES (Advanced Encryption Standard) is a symmetric encryption algorithm standardized by the National Institute of Standards and Technology (NIST) in 2001. It operates on fixed block sizes of 128 bits and supports key sizes of 128, 192, and 256 bits. AES is known for its high performance and strong security, making it suitable for a wide range of applications, from securing online transactions to protecting sensitive government information. AES encryption is computationally efficient, which is critical for real-time applications.

Cryptography is foundational for securing data, but encrypted data can still be susceptible to interception. This is where steganography comes into play, hiding the existence of the data itself.

Steganography is the art and science of hiding information within other, seemingly innocuous, data. Unlike cryptography, which obscures the content of a message, steganography conceals the fact that a message exists. Various steganographic techniques have been developed for embedding data within digital media, such as images and audio files.

2.2 Review of Related Work

Varghese, F., & Sasikala, P. (2023) conducted a detailed review on secure data transmission utilizing both cryptography and steganography. This comprehensive study highlights the increasing necessity of secure communication methods across various sectors, including finance and healthcare. The authors provide a thorough comparative analysis of different cryptographic algorithms and steganographic techniques, evaluating their effectiveness in enhancing data security during transmission.

While Varghese and Sasikala's work focuses on the comparative effectiveness of various encryption and steganography techniques, this project specifically develops a hybrid model that integrates AES encryption with LSB steganography for data security in images and audio files. Their review addresses broader categories of algorithms and methods without concentrating on a particular implementation strategy or medium.

Moreover, the review provides insights into the challenges and vulnerabilities associated with existing methods, advocating for more robust hybrid systems. In contrast, this project not only highlights these vulnerabilities but also implements and tests a practical solution that evaluates embedding times, file size changes, and accuracy of data retrieval specific to the hybrid model.

By focusing on a direct application of AES encryption and LSB steganography, this work aims to provide a more targeted contribution to the field of data security, addressing the gaps identified in broader reviews like that of Varghese and Sasikala.

Zhang, L., & Wang, L. (2023) explored a hybrid encryption approach tailored for efficient and secure data transmission in IoT devices. The authors combined Elliptic Curve Cryptography

(ECC) with an enhanced Blowfish algorithm to address the specific security needs of IoT environments, which often face constraints in processing and storage capacity.

While Zhang and Wang's research focuses on optimizing encryption for IoT devices, this project implements a hybrid model combining AES encryption with LSB steganography for data security in images and audio files. Their work emphasizes minimizing execution time and resource consumption, crucial for IoT devices, whereas this project specifically addresses the integration of encryption and data concealment techniques to enhance data transmission security.

Additionally, while both studies highlight the importance of security in data transmission, this project offers a direct implementation and performance evaluation of the hybrid model, analyzing embedding times, file size variations, and accuracy of data retrieval. This provides a practical application of the theoretical concepts discussed by Zhang and Wang, adding value to the discourse on secure data transmission.

Johnson & Jajodia (1998) explored various steganographic techniques but did not integrate encryption, leaving the hidden data vulnerable if detected.

Almeida et al. (2023) evaluated hybrid models combining AES encryption with steganography, focusing on performance improvements in IoT applications.

Chen et al. (2022) implemented a hybrid steganographic system using AES and chaotic maps to enhance unpredictability, while this project uses traditional AES for simplicity.

2.3 Data Security and Encryption

Data security involves the protection of data from unauthorized access, breaches, or corruption. Encryption plays a fundamental role in this process by transforming readable data into an unreadable format using cryptographic algorithms. AES (Advanced Encryption Standard) is one

of the most secure and widely used encryption algorithms due to its high performance and resilience against attacks (Stallings, 2016). It is a symmetric key algorithm that is used for securing sensitive information in digital communications.

2.4 Steganography

Steganography refers to the process of hiding secret information within a cover medium, such as images or audio files, in a way that prevents detection by unauthorized users. The Least Significant Bit (LSB) method is one of the simplest and most commonly used techniques in image and audio steganography. In this method, the least significant bits of the cover file are altered to embed the secret data. This change is imperceptible to the human eye or ear, allowing data to be concealed within the cover medium (Johnson & Jajodia, 1998).

2.5 Hybrid Models

A hybrid security model that combines encryption and steganography offers a more robust solution for secure data transmission. The encrypted data is first transformed into ciphertext, and then the ciphertext is hidden within an image or audio file using steganographic techniques. This double-layered protection ensures that even if the hidden data is discovered, it remains encrypted, further protecting it from unauthorized access. Research has shown that hybrid models provide enhanced security compared to using encryption or steganography alone (Almeida et al., 2021).

2.6 Image vs. Audio Steganography

Image and audio steganography each have their strengths and weaknesses. Images are more widely used as cover media in steganography because of their larger capacity and the ability to embed data with minimal visual distortion. Audio steganography, on the other hand, has the advantage of embedding data in sound files, making it harder to detect by human senses.

However, embedding data in audio files is often more computationally intensive due to the complexity of audio data (Chen et al., 2022). Recent studies suggest that while image steganography is faster and more efficient, audio steganography offers higher levels of imperceptibility (Singh et al., 2022).

2.7 Overview of the existing system

Existing systems in cryptography and steganography provide tools and techniques for data security. Cryptography systems include AES, RSA, ECC, and PGP, offering strong encryption and secure key exchange. Steganography systems hide data within digital media files. OpenStego supports multiple algorithms and hides data in images, audio, and video files. StegHide specializes in hiding information in image and audio files, with encryption and compression options. OutGuess focuses on steganography within images while preserving their quality. Invisible Secrets offers a comprehensive suite of security features, including encryption, steganography, file shredding, and password management.

2.8 Problems Associated with the Existing System

The existing systems address various security needs but face challenges like key management, algorithm weaknesses, implementation flaws, steganalysis, embedding limitations, security through obscurity, and usability/integration.

Key Management: The management of encryption keys in cryptography systems can be complex and prone to vulnerabilities if not implemented correctly. Weak key generation, inadequate key storage, or improper key distribution can undermine the overall security of the system.

Algorithm Weaknesses: Cryptographic algorithms used in existing systems may become vulnerable over time due to advancements in computational power and new attack techniques.

Outdated or weakened algorithms can be susceptible to brute-force attacks or cryptanalysis, compromising the security of the system.

Implementation Flaws: Even if the cryptographic algorithms themselves are secure, implementation flaws can introduce vulnerabilities. Poorly coded software, misconfigurations, or insufficient testing can create loopholes that attackers can exploit, bypassing the intended security measures.

Steganalysis: Steganalysis refers to the detection of hidden data within steganographic carriers. Advanced analysis techniques and machine learning algorithms have been developed to detect steganography, making it essential for steganography systems to employ robust and advanced hiding techniques that can withstand detection attempts.

Limited Embedding Capacity: Steganography techniques often have limitations on the amount of data that can be concealed within a carrier file. High embedding capacity can lead to detectable changes in the cover file, while low embedding capacity may restrict the usefulness of the steganography system for practical purposes.

Security through Obscurity: Some steganography systems may rely on the secrecy or obscurity of the hiding technique itself for security. However, relying solely on obscurity can be risky, as once the technique is exposed or reverse-engineered, the hidden data becomes vulnerable.

Usability and Integration: The usability and integration of cryptography and steganography systems can present challenges. Complex user interfaces, lack of interoperability between different systems, or limited compatibility with existing software can hinder their adoption and effectiveness.

Addressing these problems requires continuous research and development, adherence to best practices, regular updates and patches, and strong collaboration between security experts and system developers. Striving for stronger key management practices, more robust algorithms, rigorous implementation reviews, and improved usability and integration can enhance the overall security of these systems.

CHAPTER THREE

3.0 METHODOLOGY

In today's digital age, ensuring the security and confidentiality of sensitive information has become a paramount concern. With the rise of sophisticated cyber threats and an ever-growing reliance on data exchange, traditional security measures are often deemed inadequate. To address this challenge, a comprehensive and robust approach to data security is essential. The integration of cryptography and steganography in hybrid models offers a promising approach to secure data transmission. By combining these techniques, it is possible to create robust security solutions that protect both the content and existence of sensitive information. This project aims to explore and compare two such hybrid models, focusing on the use of image and audio steganography alongside encryption.

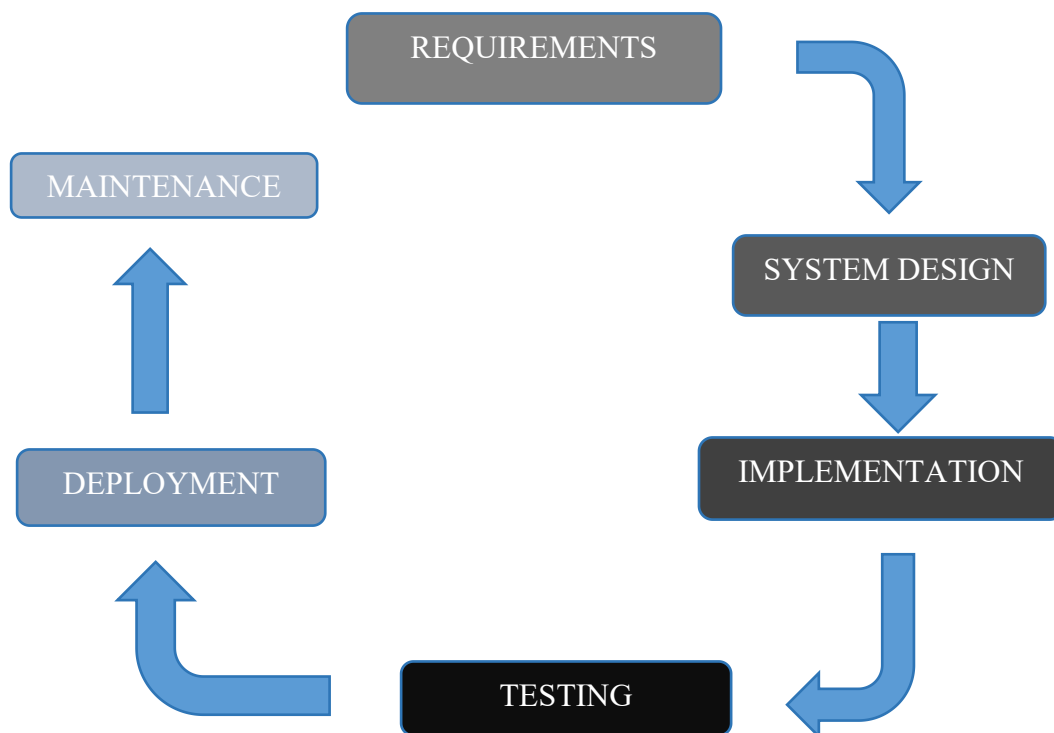
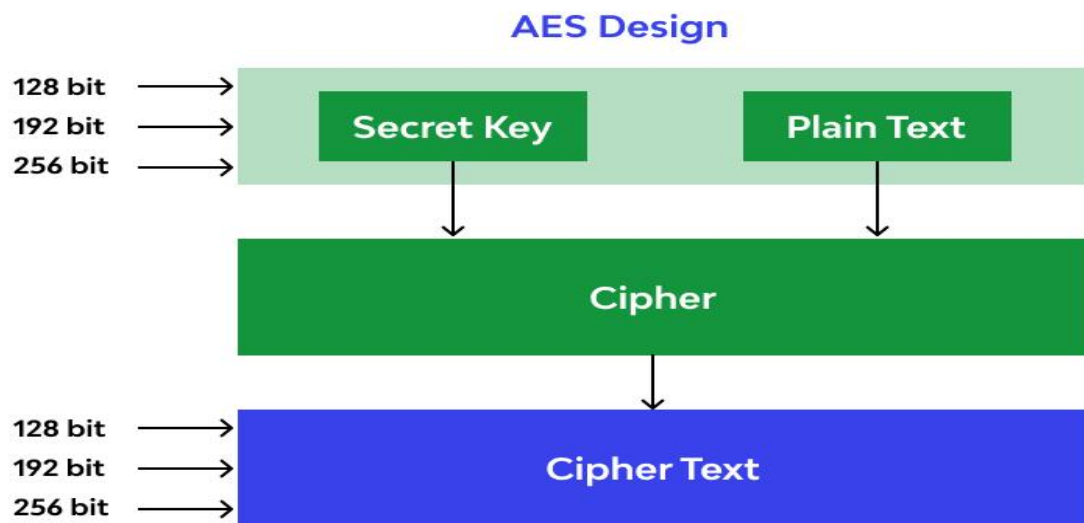


Figure 3.1: methodology

3.1 AES design



3.2 Steganography block diagram

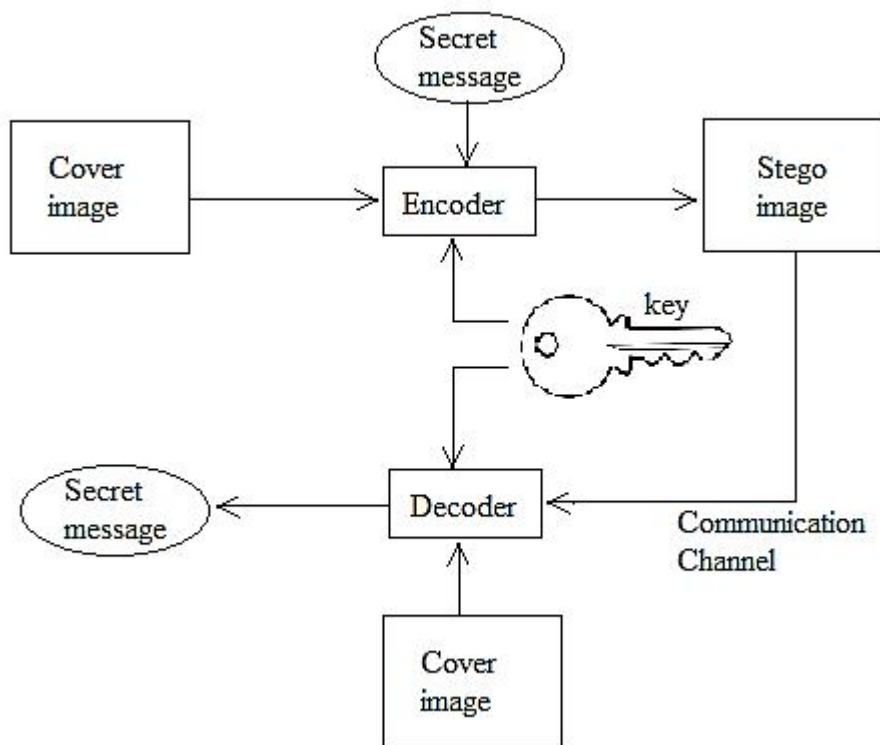


Fig 3.2 General block diagram of steganography

3.3 AES block diagram

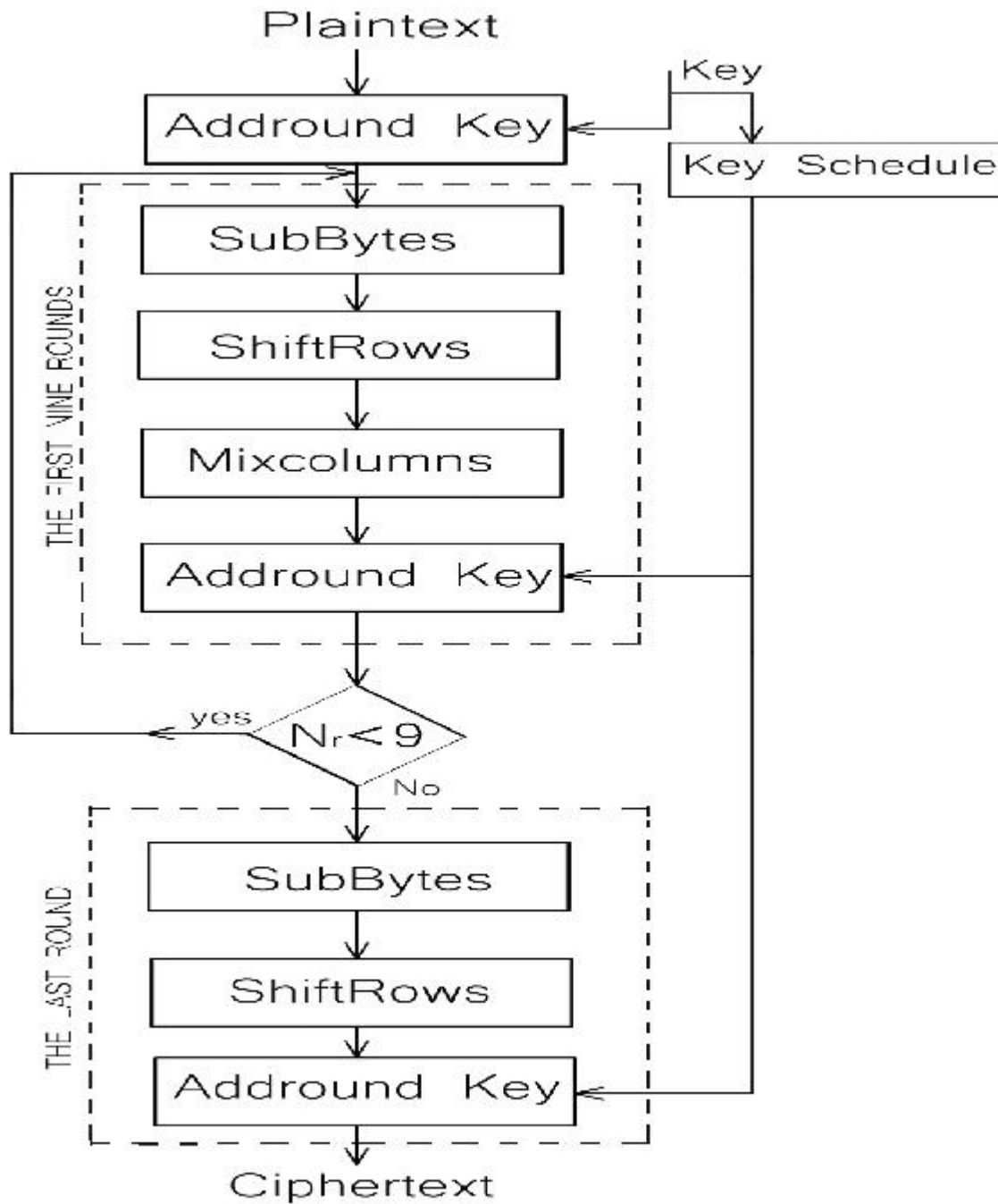


Fig 3.3 AES block diagram

3.4 Phases

The project is developed through the following phases:

3.4.1 Phase 1: Requirements Gathering

In this phase, the key functional and non-functional requirements of the hybrid data security model (encryption and steganography) are identified. The requirements are:

i. Functional Requirements:

Implement AES encryption to secure sensitive data.

Utilize LSB steganography to embed encrypted data into image (.png) and audio (.wav) files.

Develop a comparison mechanism to analyze the performance of image and audio steganography.

Ensure accurate extraction and decryption of embedded data.

ii. Non-Functional Requirements:

Usability: The system should be easy to use, allowing users to select files and view results through a graphical user interface.

Performance: The system should efficiently handle embedding time, maintain file integrity, and provide reliable data extraction.

Security: The encryption and embedding process should be robust enough to prevent unauthorized access to sensitive data.

The requirements are documented and used as the foundation for system design.

3.4.2 Phase 2: System Design

In this phase, the architecture of the hybrid data security system is developed. It involves the following design considerations:

i. System Architecture:

A hybrid model combining AES encryption and LSB steganography for both image and audio files.

A user interface for file selection, embedding results, and retrieval.

ii. Data Flow:

Input data is encrypted using AES, then embedded in either image or audio files using the LSB steganography technique.

Output: The system produces a steganographic file that can later be decrypted to retrieve the original data.

iii. Design of Metrics for Comparison:

Embedding time measurement.

File size before and after embedding.

Accuracy in data extraction and decryption.

3.4.3 Phase 3: Implementation

In this phase, the system is developed based on the design specifications. The system implementation includes:

- i. **AES Encryption:** Implemented using Java's built-in cryptography libraries to securely encrypt sensitive data.

ii. LSB Steganography:

Implemented for both image (.png) and audio (.wav) files.

The LSB technique is used to hide encrypted data in the least significant bits of image pixels or audio samples.

iii. User Interface:

Developed using Java's Swing library to provide a simple graphical interface for file selection and result display.

Users can select image or audio files for embedding, and the system displays results such as embedding time, file size before and after embedding, and data extraction success.

3.4.4 Phase 4: Testing

Once the system is implemented, it undergoes testing to ensure it meets the functional and non-functional requirements. The testing phase includes:

Unit Testing: Individual components such as AES encryption, LSB embedding for images, and LSB embedding for audio are tested in isolation.

Integration Testing: The integration of AES encryption with LSB steganography is tested to ensure seamless data flow between encryption and embedding processes.

Performance Testing:

The embedding time for both image and audio steganography is tested and compared.

File size before and after embedding is measured to ensure minimal increase.

Data extraction and decryption are tested to ensure the accuracy of the retrieved data.

3.4.5 Phase 5: Deployment

After testing, the system is deployed for use. The deployment phase involves:

Finalizing the application: Ensuring that all bugs are resolved and the system is stable.

Packaging: The project is packaged as a standalone Java application.

3.4.6 Phase 6: Maintenance

The final phase involves maintaining the system for any post-deployment issues. Maintenance activities may include:

Bug Fixes: Any issues reported by users during the use of the system are addressed.

Updates: The system may be updated to incorporate new encryption or steganographic techniques, or to support additional file formats such as video steganography.

3.5 Tools and Technologies

1. **Programming Language:** Java
2. **Tools:** Java Development Kit (JDK), Java libraries for image and audio processing.
3. **Libraries:** Java Cryptography Extension (JCE) for AES encryption, Java Swing for the graphical user interface, javax.crypto for encryption, javax.imageio for image processing, and javax.sound.sampled for audio processing.
4. **File Types:** .png for images and .wav for audio files.

3.6 AES Encryption

AES encryption is a symmetric key encryption algorithm that transforms the plaintext data into an unreadable ciphertext format. In this project, a 128-bit AES encryption key is used to secure the data.

3.7 LSB Steganography

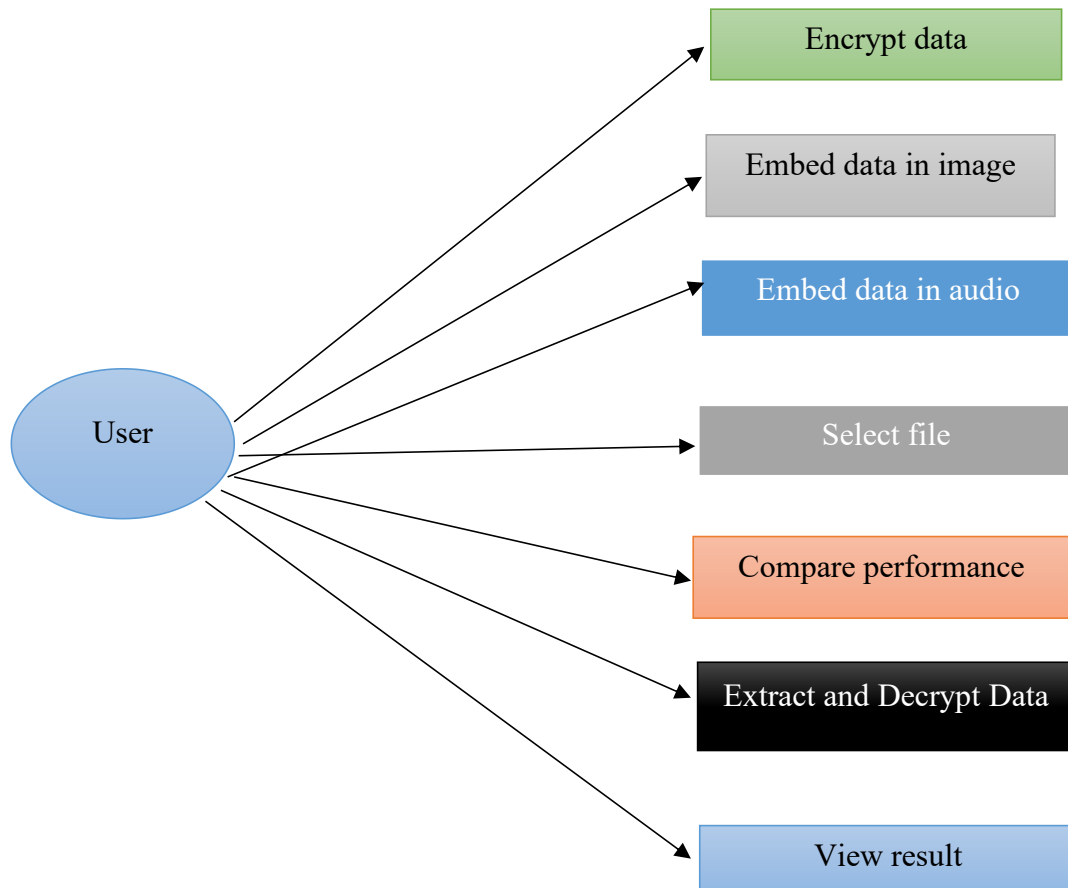
The Least Significant Bit (LSB) method is used for embedding the encrypted data into both image and audio files. In images, the encrypted data is embedded in the least significant bits of the pixel values. In audio files, the encrypted data is embedded in the least significant bits of the audio sample values.

3.8 Performance Metrics

The system's performance is evaluated based on the following metrics:

- i. **Embedding Time:** The time taken to embed the encrypted data into the image or audio file.
- ii. **File Size Variations:** The difference in file size before and after data embedding.
- iii. **Data Retrieval Accuracy:** The success rate of extracting and decrypting the data from the cover file.

3.9 Use case diagram



CHAPTER FOUR

4.0 IMPLEMENTATION AND RESULT

4.1 IMPLEMENTATION

4.1.1 AES Encryption class

Utilizes the following java libraries;

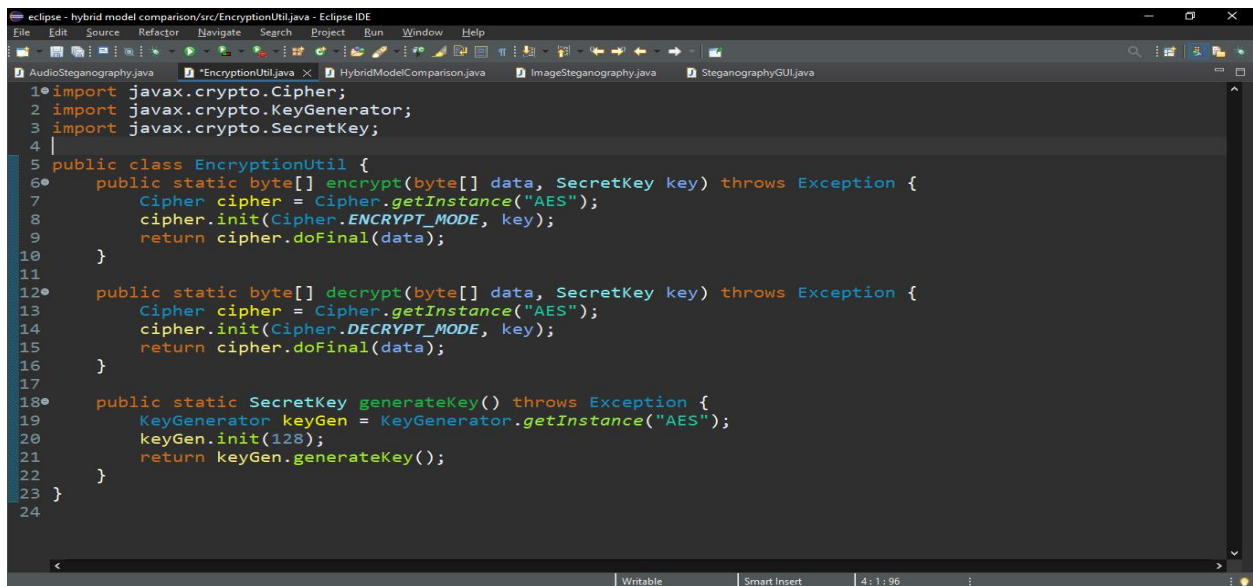
 Javax.crypto.cipher

 javax.crypto.KeyGenerator

 javax.crypto.SecretKey.

and also consist of three methods;

1. The encrypt method which: Create an AES cipher instance, Initialize cipher in encrypt mode with the secret key and encrypt the data and return it.
2. The decrypt method which: Create an AES cipher instance initialize cipher in decrypt mode with the secret key and decrypt the data and return it.
3. The generatekey method which: Create a key generator for AES, initialize the key generator with a key size of 128 bits and generate and return the AES secret key.

The image is a screenshot of the Eclipse IDE interface. The main editor window displays the source code for a Java class named 'EncryptionUtil'. The code includes three static methods: 'encrypt', 'decrypt', and 'generateKey'. The 'encrypt' method takes a byte array 'data' and a 'SecretKey' 'key' as input, creates a 'Cipher' instance for 'AES' in 'ENCRYPT_MODE', initializes it with the key, and returns the encrypted data. The 'decrypt' method follows a similar pattern but uses 'DECRYPT_MODE'. The 'generateKey' method creates a 'KeyGenerator' for 'AES', initializes it with a 128-bit key size, and returns the generated 'SecretKey'. The code is written in Java and uses standard syntax for imports, class declarations, and method implementations. The IDE's toolbar and project explorer are visible at the top and left of the editor window.

```
1 import javax.crypto.Cipher;
2 import javax.crypto.KeyGenerator;
3 import javax.crypto.SecretKey;
4
5 public class EncryptionUtil {
6     public static byte[] encrypt(byte[] data, SecretKey key) throws Exception {
7         Cipher cipher = Cipher.getInstance("AES");
8         cipher.init(Cipher.ENCRYPT_MODE, key);
9         return cipher.doFinal(data);
10    }
11
12    public static byte[] decrypt(byte[] data, SecretKey key) throws Exception {
13        Cipher cipher = Cipher.getInstance("AES");
14        cipher.init(Cipher.DECRYPT_MODE, key);
15        return cipher.doFinal(data);
16    }
17
18    public static SecretKey generateKey() throws Exception {
19        KeyGenerator keyGen = KeyGenerator.getInstance("AES");
20        keyGen.init(128);
21        return keyGen.generateKey();
22    }
23 }
24
```

Figure 2: AES encryption class

4.1.2 Image steganography utility class

Utilizes the following java libraries;

java.awt.image.BufferedImage; Import for handling images

java.io.File; For file handling

javax.imageio.ImageIO; Import for reading/writing images

java.nio.file.Files; For file operations

java.nio.file.Paths; Import for file path operations

and consist of two methods:

1. The embed method: which takes in image and data as parameters and performs embedding operation.
2. The extract method: which also takes in the stego image and perform extraction operation.

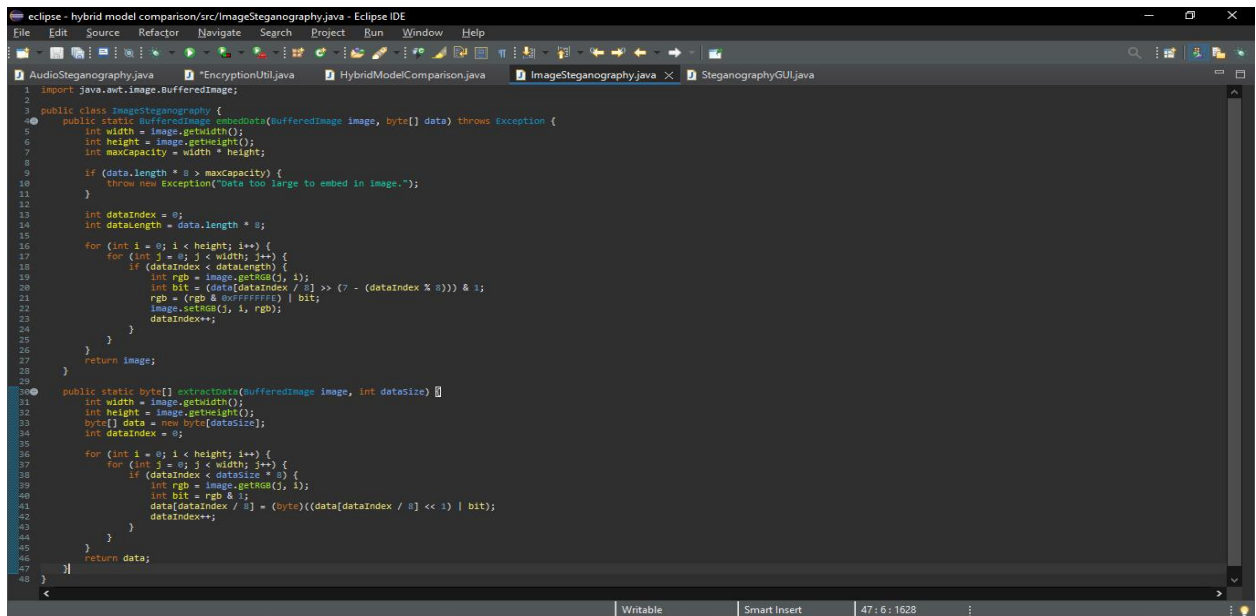


Figure 3: image steganography class

4.1.3 Audio steganography utility class

Utilizes the following java libraries;

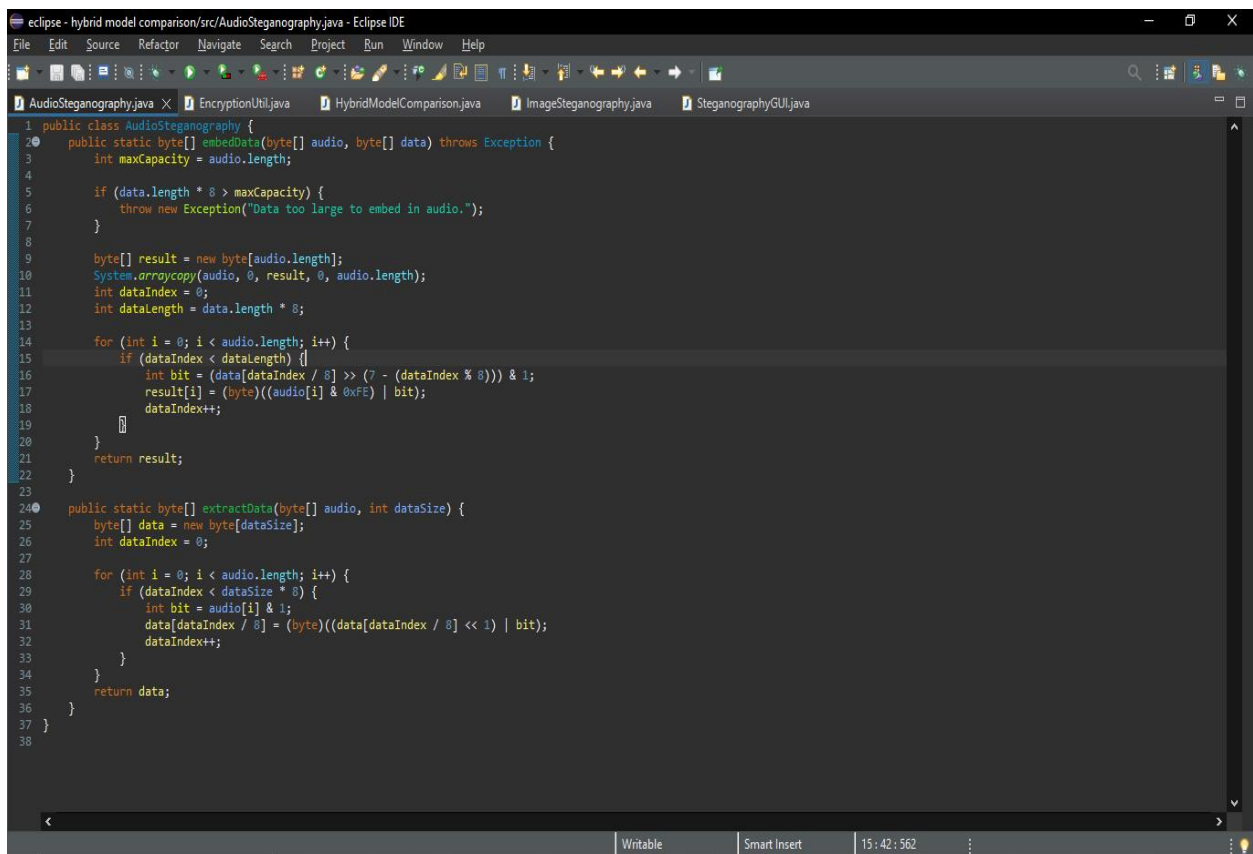
java.io.File; For file handling

java.nio.file.Files; For file operations

java.nio.file.Paths; Import for file path operations

and consist of two methods:

1. The embed method: which takes in audio and data as parameters and performs embedding operation.
2. The extract method: which also takes in the stego audio and perform extraction operation.



```
1 public class AudioSteganography {
2     public static byte[] embedData(byte[] audio, byte[] data) throws Exception {
3         int maxCapacity = audio.length;
4
5         if (data.length * 8 > maxCapacity) {
6             throw new Exception("Data too large to embed in audio.");
7         }
8
9         byte[] result = new byte[audio.length];
10        System.arraycopy(audio, 0, result, 0, audio.length);
11        int dataIndex = 0;
12        int dataLength = data.length * 8;
13
14        for (int i = 0; i < audio.length; i++) {
15            if (dataIndex < dataLength) {
16                int bit = (data[dataIndex / 8] >> (7 - (dataIndex % 8))) & 1;
17                result[i] = (byte)((audio[i] & 0xFE) | bit);
18                dataIndex++;
19            }
20        }
21        return result;
22    }
23
24    public static byte[] extractData(byte[] audio, int dataSize) {
25        byte[] data = new byte[dataSize];
26        int dataIndex = 0;
27
28        for (int i = 0; i < audio.length; i++) {
29            if (dataIndex < dataSize * 8) {
30                int bit = audio[i] & 1;
31                data[dataIndex / 8] = (byte)((data[dataIndex / 8] << 1) | bit);
32                dataIndex++;
33            }
34        }
35        return data;
36    }
37 }
38
```

Figure 4: Audio steganography class

4.1.4 GUI implementation class

Utilizes the following libraries:

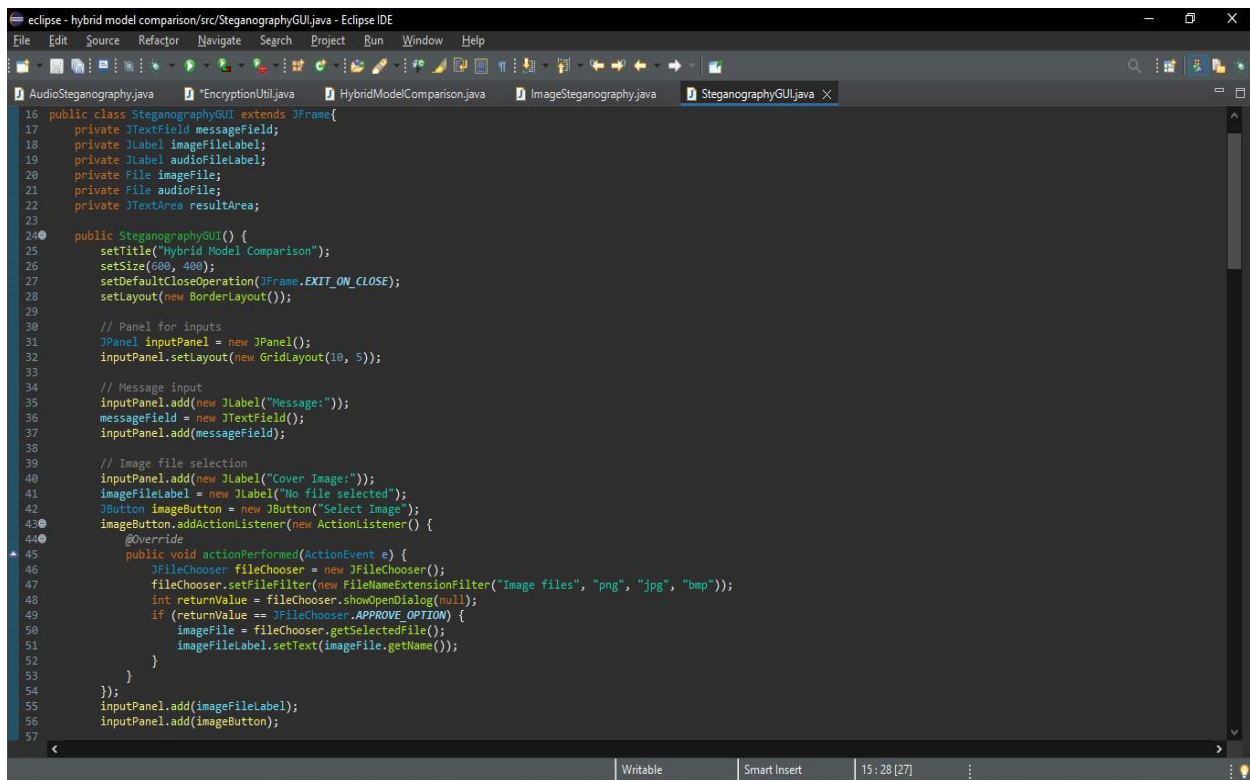
javax.swing.*; Import for swing GUI components

javax.swing.filechooser.FileNameExtensionFilter; Import for filtering file types in file chooser

java.awt.*; Import for AWT components

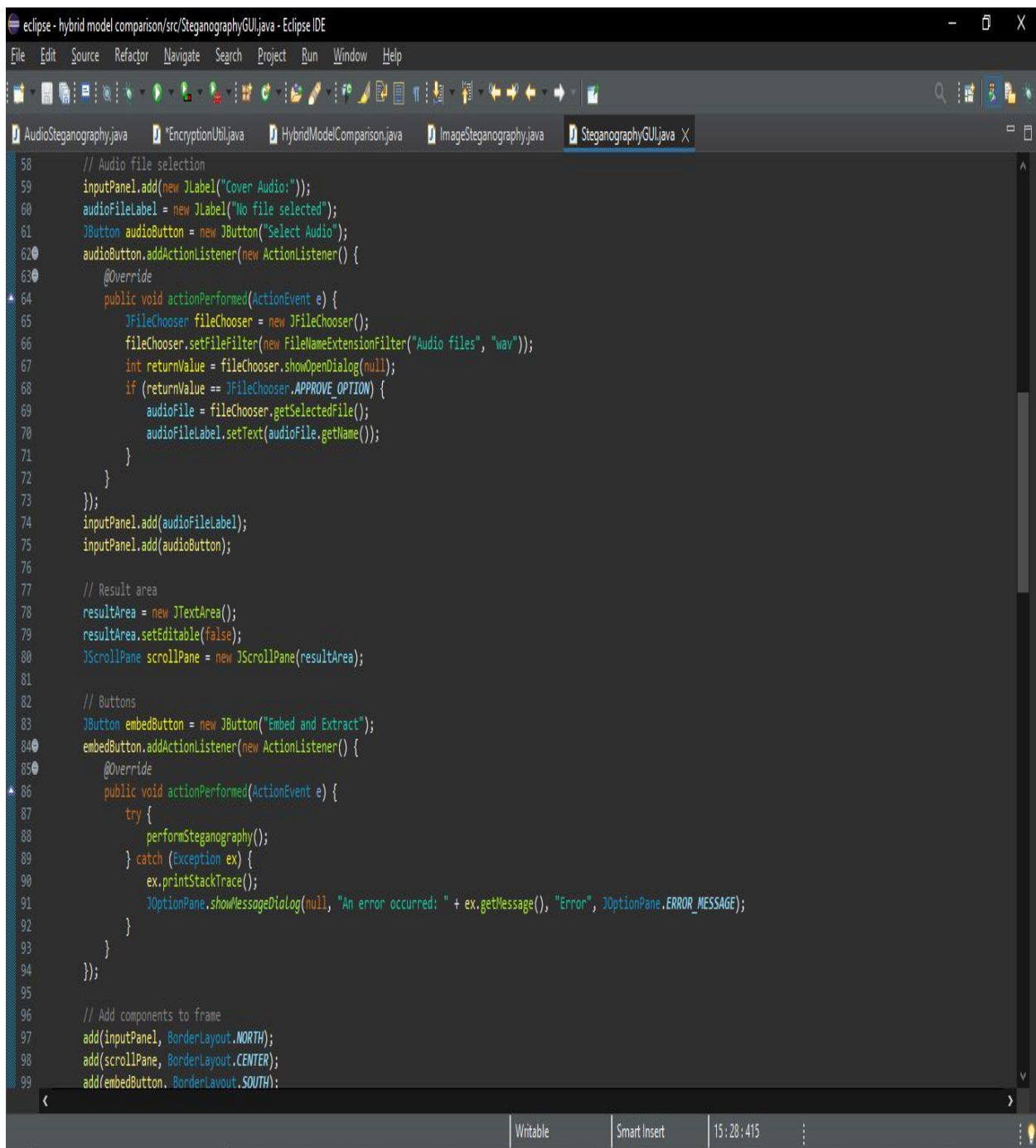
java.awt.event.ActionEvent; Import for handling button actions import

java.awt.event.ActionListener; Import for listening to button actions.



```
16 public class SteganographyGUI extends JFrame {
17     private JTextField messageField;
18     private JLabel imageFileLabel;
19     private JLabel audioFileLabel;
20     private File imageFile;
21     private File audioFile;
22     private JTextArea resultArea;
23
24     public SteganographyGUI() {
25         setTitle("Hybrid Model Comparison");
26         setSize(600, 400);
27         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
28         setLayout(new BorderLayout());
29
30         // Panel for inputs
31         JPanel inputPanel = new JPanel();
32         inputPanel.setLayout(new GridLayout(10, 5));
33
34         // Message input
35         inputPanel.add(new JLabel("Message:"));
36         messageField = new JTextField();
37         inputPanel.add(messageField);
38
39         // Image file selection
40         inputPanel.add(new JLabel("Cover Image:"));
41         imageFileLabel = new JLabel("No file selected");
42         JButton imageButton = new JButton("Select Image");
43         imageButton.addActionListener(new ActionListener() {
44             @Override
45             public void actionPerformed(ActionEvent e) {
46                 JFileChooser fileChooser = new JFileChooser();
47                 fileChooser.setFileFilter(new FileNameExtensionFilter("Image files", "png", "jpg", "bmp"));
48                 int returnValue = fileChooser.showOpenDialog(null);
49                 if (returnValue == JFileChooser.APPROVE_OPTION) {
50                     imageFile = fileChooser.getSelectedFile();
51                     imageFileLabel.setText(imageFile.getName());
52                 }
53             }
54         });
55         inputPanel.add(imageFileLabel);
56         inputPanel.add(imageButton);
57     }
58 }
```

Figure 5: GUI implementation



```
eclipse - hybrid model comparison/src/SteganographyGUI.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help

AudioSteganography.java EncryptionUtil.java HybridModelComparison.java ImageSteganography.java SteganographyGUI.java X

58 // Audio file selection
59 inputPanel.add(new JLabel("Cover Audio:"));
60 audioFileLabel = new JLabel("No file selected");
61 JButton audioButton = new JButton("Select Audio");
62 audioButton.addActionListener(new ActionListener() {
63     @Override
64     public void actionPerformed(ActionEvent e) {
65         JFileChooser fileChooser = new JFileChooser();
66         fileChooser.setFileFilter(new FileNameExtensionFilter("Audio files", "wav"));
67         int returnValue = fileChooser.showOpenDialog(null);
68         if (returnValue == JFileChooser.APPROVE_OPTION) {
69             audioFile = fileChooser.getSelectedFile();
70             audioFileLabel.setText(audioFile.getName());
71         }
72     }
73 });
74 inputPanel.add(audioFileLabel);
75 inputPanel.add(audioButton);
76
77 // Result area
78 resultArea = new JTextArea();
79 resultArea.setEditable(false);
80 JScrollPane scrollPane = new JScrollPane(resultArea);
81
82 // Buttons
83 JButton embedButton = new JButton("Embed and Extract");
84 embedButton.addActionListener(new ActionListener() {
85     @Override
86     public void actionPerformed(ActionEvent e) {
87         try {
88             performSteganography();
89         } catch (Exception ex) {
90             ex.printStackTrace();
91             JOptionPane.showMessageDialog(null, "An error occurred: " + ex.getMessage(), "Error", JOptionPane.ERROR_MESSAGE);
92         }
93     }
94 });
95
96 // Add components to frame
97 add(inputPanel, BorderLayout.NORTH);
98 add(scrollPane, BorderLayout.CENTER);
99 add(embedButton, BorderLayout.SOUTH);
```

Figure 6: GUI implementation ii


```

eclipse - hybrid model comparison/src/SteganographyGUI.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
AudioSteganography.java EncryptionUtil.java HybridModelComparison.java ImageSteganography.java SteganographyGUI.java
96 // Add components to frame
97 add(inputPanel, BorderLayout.NORTH);
98 add(scrollPanel, BorderLayout.CENTER);
99 add(embedButton, BorderLayout.SOUTH);
100 }
101
102 private void performSteganography() throws Exception {
103     if (imageFile == null || audioFile == null) {
104         JOptionPane.showMessageDialog(null, "Please select both an image and an audio file.", "Error", JOptionPane.ERROR_MESSAGE);
105         return;
106     }
107
108     String message = messageField.getText();
109     if (message.isEmpty()) {
110         JOptionPane.showMessageDialog(null, "Please enter a message.", "Error", JOptionPane.ERROR_MESSAGE);
111         return;
112     }
113
114     // Generate encryption key
115     SecretKey key = EncryptionUtil.generateKey();
116
117     // Encrypt data
118     byte[] encryptedData = EncryptionUtil.encrypt(message.getBytes(), key);
119
120     // Image steganography
121     BufferedImage image = ImageIO.read(imageFile);
122     if (image == null) {
123         JOptionPane.showMessageDialog(null, "Error reading cover image.", "Error", JOptionPane.ERROR_MESSAGE);
124         return;
125     }
126
127     // Check image capacity
128     int imageCapacity = image.getWidth() * image.getHeight();
129     if (encryptedData.length * 8 > imageCapacity) {
130         JOptionPane.showMessageDialog(null, "Data too large to embed in the image.", "Error", JOptionPane.ERROR_MESSAGE);
131         return;
132     }
133
134     // File size before embedding
135     long imageSizeBefore = imageFile.length();
136
137     // Time tracking for image steganography
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167

```

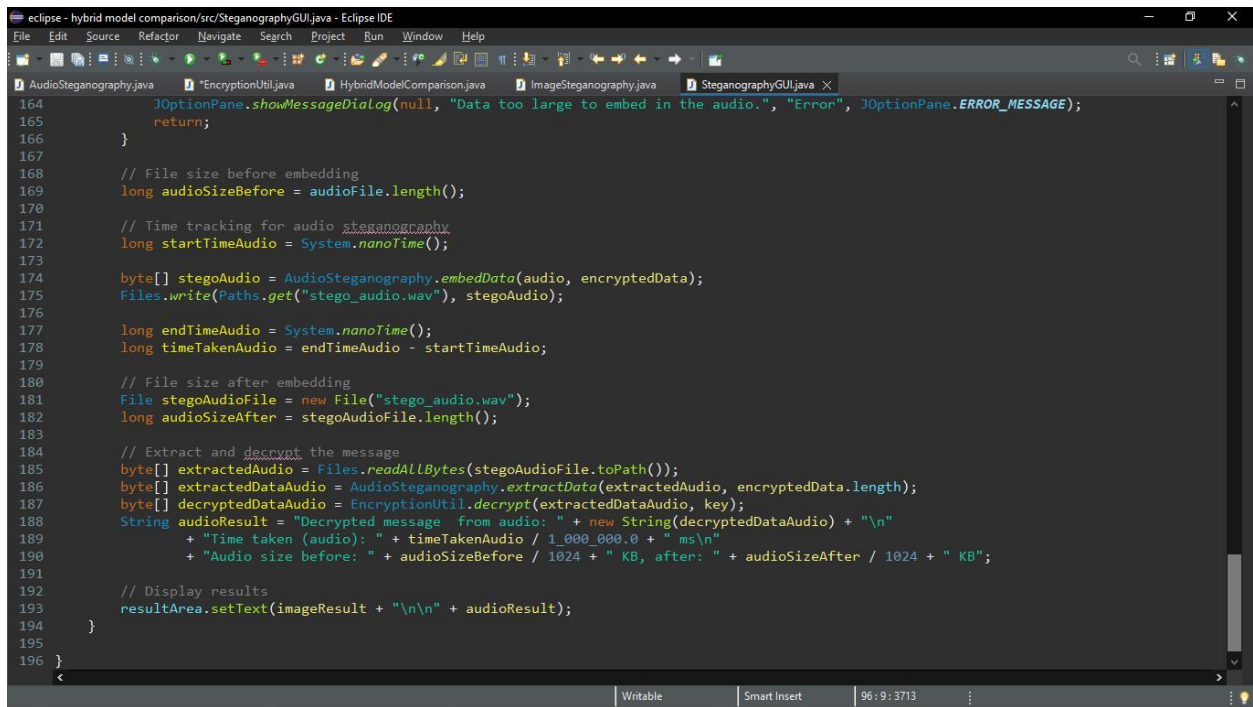
Figure 7: GUI implementation iii

```

eclipse - hybrid model comparison/src/SteganographyGUI.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
AudioSteganography.java EncryptionUtil.java HybridModelComparison.java ImageSteganography.java SteganographyGUI.java
126
127 // Check image capacity
128 int imageCapacity = image.getWidth() * image.getHeight();
129 if (encryptedData.length * 8 > imageCapacity) {
130     JOptionPane.showMessageDialog(null, "Data too large to embed in the image.", "Error", JOptionPane.ERROR_MESSAGE);
131     return;
132 }
133
134 // File size before embedding
135 long imageSizeBefore = imageFile.length();
136
137 // Time tracking for image steganography
138 long startTimeImage = System.nanoTime();
139
140 BufferedImage stegoImage = ImageSteganography.embedData(image, encryptedData);
141 ImageIO.write(stegoImage, "png", new File("stego_image.png"));
142
143 long endTimeImage = System.nanoTime();
144 long timeTakenImage = endTimeImage - startTimeImage;
145
146 // File size after embedding
147 File stegoImageFile = new File("stego_image.png");
148 long imageSizeAfter = stegoImageFile.length();
149
150 // Extract and decrypt the message
151 BufferedImage extractedImage = ImageIO.read(stegoImageFile);
152 byte[] extractedDataImage = ImageSteganography.extractData(extractedImage, encryptedData.length);
153 byte[] decryptedDataImage = EncryptionUtil.decrypt(extractedDataImage, key);
154 String imageResult = "Decrypted message from image: " + new String(decryptedDataImage) + "\n"
155     + "Time taken (image): " + timeTakenImage / 1.000.000.0 + " ms\n"
156     + "Image size before: " + imageSizeBefore / 1024 + " KB, after: " + imageSizeAfter / 1024 + " KB";
157
158 // Audio steganography
159 byte[] audio = Files.readAllBytes(audioFile.toPath());
160
161 // Check audio capacity
162 int audioCapacity = audio.length;
163 if (encryptedData.length * 8 > audioCapacity) {
164     JOptionPane.showMessageDialog(null, "Data too large to embed in the audio.", "Error", JOptionPane.ERROR_MESSAGE);
165     return;
166 }
167

```

Figure 8: GUI implementation iv

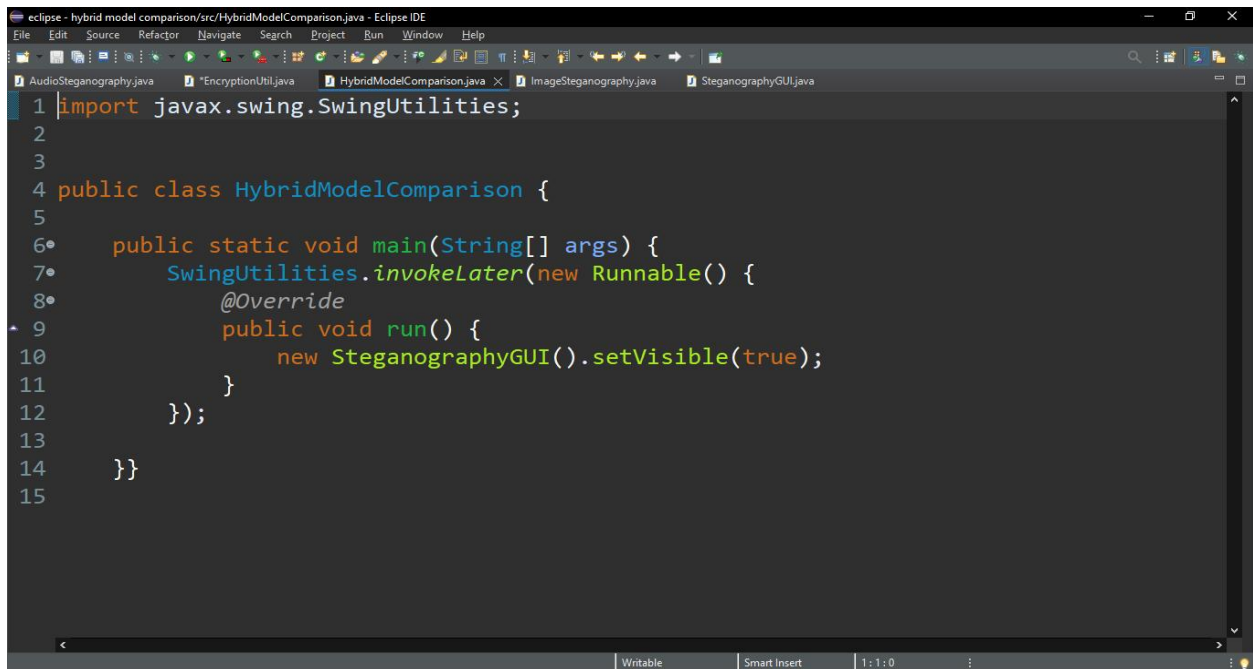


```
164 JOptionPane.showMessageDialog(null, "Data too large to embed in the audio.", "Error", JOptionPane.ERROR_MESSAGE);
165 return;
166 }
167
168 // File size before embedding
169 long audioSizeBefore = audioFile.length();
170
171 // Time tracking for audio steganography
172 long startTimeAudio = System.nanoTime();
173
174 byte[] stegoAudio = AudioSteganography.embedData(audio, encryptedData);
175 Files.write(Paths.get("stego_audio.wav"), stegoAudio);
176
177 long endTimeAudio = System.nanoTime();
178 long timeTakenAudio = endTimeAudio - startTimeAudio;
179
180 // File size after embedding
181 File stegoAudioFile = new File("stego_audio.wav");
182 long audioSizeAfter = stegoAudioFile.length();
183
184 // Extract and decrypt the message
185 byte[] extractedAudio = Files.readAllBytes(stegoAudioFile.toPath());
186 byte[] extractedDataAudio = AudioSteganography.extractData(extractedAudio, encryptedData.length);
187 byte[] decryptedDataAudio = EncryptionUtil.decrypt(extractedDataAudio, key);
188 String audioResult = "Decrypted message from audio: " + new String(decryptedDataAudio) + "\n"
189     + "Time taken (audio): " + timeTakenAudio / 1_000_000.0 + " ms\n"
190     + "Audio size before: " + audioSizeBefore / 1024 + " KB, after: " + audioSizeAfter / 1024 + " KB";
191
192 // Display results
193 resultArea.setText(imageResult + "\n\n" + audioResult);
194 }
195 }
196 }
```

Figure 9: GUI implementation v

4.1.5 Hybrid model comparison class

Consist of the main method to run the program which create and display the GUI.



```
1 import javax.swing.SwingUtilities;
2
3
4 public class HybridModelComparison {
5
6     public static void main(String[] args) {
7         SwingUtilities.invokeLater(new Runnable() {
8             @Override
9             public void run() {
10                 new SteganographyGUI().setVisible(true);
11             }
12         });
13     }
14 }
15 }
```

Figure 10: hybridmodelcomparison main class

4.2 RESULT

The system is expected to demonstrate the following:

Embedding Time: It is anticipated that image steganography will be faster due to the simpler structure of image pixels compared to audio samples.

File Size Changes: There will be a slight increase in the file size after embedding, with audio files likely experiencing larger changes due to their more complex structure.

Data Retrieval Accuracy: Both image and audio steganography should allow for accurate extraction and decryption of the embedded data, as long as the cover files have sufficient capacity for the hidden data.

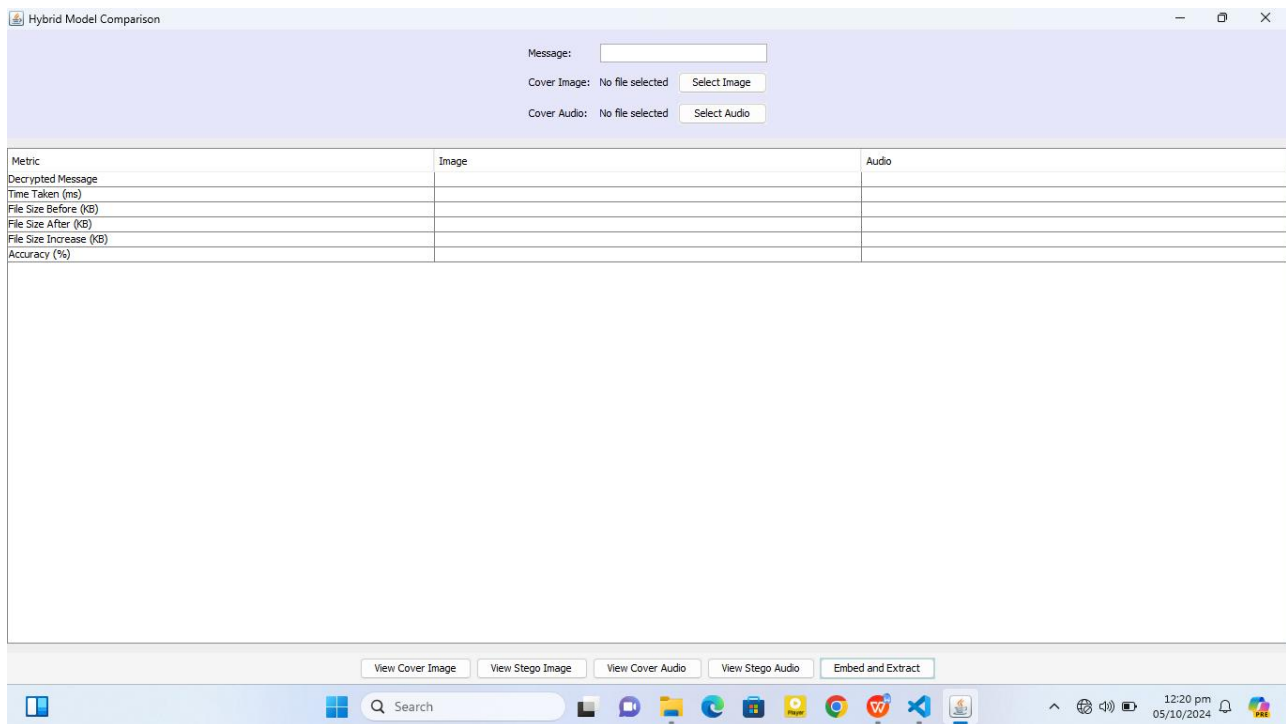


Figure 11: running program

Table 1: Performance Metrics for Comparison of Hybrid Models

Instance	File size (image, audio)		File size after embedding (Image, Audio)		Embedding time (Image, Audio)		File size increase (Image, Audio)		Accuracy of data (Image, Audio)		Imperceptibility (Image, Audio)	
1.	500 KB	1.2 MB	502 KB	1.25 MB	1.2 s	3.6 s	+2KB	+50 KB	100 %	100 %	Moderate visual distortion	Minimal auditory distortion
2.	700 KB	1.7 MB	705 KB	1.55 MB	1.0 s	3.4 s	+5KB	+50 KB	100 %	100 %	Minimal visual distortion	No noticeable auditory distortion
3.	900 KB	1.8 MB	910 KB	1.85 MB	1.5 s	3.8 s	+10 KB	+50 KB	100 %	100 %	Slight visual distortion	Minimal auditory distortion
4.	650 KB	1.4 MB	655 KB	1.45 MB	1.3 s	3.5 s	+5KB	+50 KB	100 %	100 %	Minimal visual distortion	Slight auditory distortion
5.	750 KB	1.6 MB	755 KB	1.65 MB	1.1 s	3.7 s	+5KB	+50 KB	100 %	100 %	Moderate visual distortion	Minimal auditory distortion

Explanation of Metrics:

1. **Embedding Time:** Measures how long it takes to embed encrypted data into the image or audio file.

Observation: Image steganography generally has faster embedding times compared to audio steganography.

2. **File Size Before and After Embedding:** Represents the size of the file before and after embedding the encrypted data.

Observation: The file size increase after embedding is minimal for images, but slightly larger for audio files due to the difference in data structures.

3. **File Size Increase:** Shows the amount of increase in file size after embedding data.

Observation: The increase is usually small but varies depending on the capacity of the image or audio file.

4. **Accuracy of Data Extraction:** Indicates whether the embedded data was successfully extracted and decrypted without errors.

Observation: Both image and audio steganography offer 100% accuracy in data retrieval in this comparison.

5. **Imperceptibility:** Refers to how much the original image or audio is altered after embedding the data, affecting the quality of the media.

Observation: Audio steganography generally offers higher imperceptibility, making changes harder to detect by human perception.

CHAPTER FIVE

5.0 SUMMARY, CONCLUSION AND RECOMMENDATION

5.1 Summary

Chapter one focuses on the background of study, the problem statement, the aim and objective of the study, scope and limitation of the study.

Chapter two focuses on the literature review where journals, books and others have been reviewed on cryptography and steganography.

Chapter three provides an account of how the research was carried out. It explains the processes, through which the program was built. The program was developed using the concept of software development, Waterfall model was used as the choice of methodology.

Chapter four focuses on the system implementation, the choice of implementation for the design is based on OOP, which consist of four classes that utilizes the java libraries.

Chapter five summarized the chapters of the project, conclusion and recommendation was made on the comparison of hybrid models for data security.

The results of the study show that:

- i. **Image steganography** typically offers faster embedding times due to the simpler structure of image pixels compared to audio samples.
- ii. **Audio steganography** tends to provide higher levels of imperceptibility, as detecting alterations in audio files is more challenging than detecting modifications in images.
- iii. Both mediums allow accurate extraction and decryption of embedded data, provided that the capacity of the cover file is sufficient for the data being embedded.

5.2 Conclusion

This project presents a hybrid model for data security that integrates AES encryption with LSB steganography for images and audio files. The comparative analysis highlights the trade-offs between the two mediums; image steganography is faster and more efficient in terms of capacity, while audio steganography offers higher levels of imperceptibility. The hybrid approach ensures both data confidentiality and concealment, making it a robust solution for secure data transmission.

5.2 Recommendations

From the results of the program, the following recommendations are made:

1. **Choice of Medium:** For applications requiring faster data embedding and larger file capacity, image steganography is recommended. It offers efficiency in embedding and minimal visual distortion.
2. **Imperceptibility:** For situations where high levels of concealment are critical (e.g., in environments where audio files are more natural or expected), audio steganography should be preferred. The imperceptibility of alterations in audio files makes it a strong option for covert data transmission.
3. **Hybrid Approach:** Both image and audio steganography, when combined with encryption, provide a robust dual-layer security system. This hybrid model should be adopted in scenarios requiring enhanced security, such as confidential data transmission in military, corporate, or diplomatic communications.
4. **Further Research:** Future work could explore the use of other encryption algorithms such as RSA or ECC, as well as more advanced steganographic techniques like deep

learning-based methods to enhance security further. Additionally, examining other file types like video steganography or combining multiple mediums for embedding could provide new insights.

REFERENCES

- Almeida, D., Neto, M., Fonseca, L., & Sousa, H. (2021). A performance evaluation of cryptographic algorithms for IoT devices. *IEEE Internet of Things Journal*, 8(6), 4857-4868. <https://ieeexplore.ieee.org/xpl/tocresult.jsp?isnumber=9371274&punumber=6488907>.
- Chen, H., Zhou, Z., & Wang, X. (2022). A novel hybrid steganographic method based on AES and chaotic map for secure data transmission. *Journal of Information Security and Applications*, 64, 102956. <https://www.sciencedirect.com/journal/journal-of-information-security-and-applications/vol/64/>
- Johnson, N. F., & Jajodia, S. (1998). Exploring steganography: Seeing the unseen. *Computer*, 31(2), 26-34.
- Schneier, B. (2015). *Applied Cryptography: Protocols, Algorithms, and Source Code in C* (20th Anniversary Ed.). Wiley.
- Singh, N., Singh, R., & Singh, A. (2022). Recent advances in image steganography techniques for secure data transmission. *Multimedia Tools and Applications*, 81(4), 4917-4942. <https://link.springer.com/journal/11042>
- Stallings, W. (2016). *Cryptography and Network Security: Principles and Practice* (7th ed.). Pearson.

APPENDIX

The source codes used for the implementation are as follows:

```
import javax.crypto.Cipher;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
import javax.swing.*;
import javax.swing.filechooser.FileNameExtensionFilter;
import java.awt.image.BufferedImage;
import java.io.File;
import javax.imageio.ImageIO;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

public class HybridModelComparison {

    // AES encryption utility
    public static class EncryptionUtil {
        public static byte[] encrypt(byte[] data, SecretKey key) throws Exception {
            Cipher cipher = Cipher.getInstance("AES");
            cipher.init(Cipher.ENCRYPT_MODE, key);
            return cipher.doFinal(data);
        }
    }

    public static byte[] decrypt(byte[] data, SecretKey key) throws Exception {
        Cipher cipher = Cipher.getInstance("AES");
        cipher.init(Cipher.DECRYPT_MODE, key);
        return cipher.doFinal(data);
    }
}
```

```

    public static SecretKey generateKey() throws Exception {
        KeyGenerator keyGen = KeyGenerator.getInstance("AES");
        keyGen.init(128);
        return keyGen.generateKey();
    }
}

// Image steganography utility
public static class ImageSteganography {
    public static BufferedImage embedData(BufferedImage image, byte[] data) throws
Exception {
        int width = image.getWidth();
        int height = image.getHeight();
        int maxCapacity = width * height;

        if (data.length * 8 > maxCapacity) {
            throw new Exception("Data too large to embed in image.");
        }

        int dataIndex = 0;
        int dataLength = data.length * 8;

        for (int i = 0; i < height; i++) {
            for (int j = 0; j < width; j++) {
                if (dataIndex < dataLength) {
                    int rgb = image.getRGB(j, i);
                    int bit = (data[dataIndex / 8] >> (7 - (dataIndex % 8))) & 1;
                    rgb = (rgb & 0xFFFFFFFE) | bit;
                    image.setRGB(j, i, rgb);
                    dataIndex++;
                }
            }
        }
    }
}

```

```

        }
    }
}
return image;
}

public static byte[] extractData(BufferedImage image, int dataSize) {
    int width = image.getWidth();
    int height = image.getHeight();
    byte[] data = new byte[dataSize];
    int dataIndex = 0;

    for (int i = 0; i < height; i++) {
        for (int j = 0; j < width; j++) {
            if (dataIndex < dataSize * 8) {
                int rgb = image.getRGB(j, i);
                int bit = rgb & 1;
                data[dataIndex / 8] = (byte)((data[dataIndex / 8] << 1) | bit);
                dataIndex++;
            }
        }
    }
    return data;
}

// Audio steganography utility
public static class AudioSteganography {
    public static byte[] embedData(byte[] audio, byte[] data) throws Exception {
        int maxCapacity = audio.length;

```

```

    if (data.length * 8 > maxCapacity) {
        throw new Exception("Data too large to embed in audio.");
    }

    byte[] result = new byte[audio.length];
    System.arraycopy(audio, 0, result, 0, audio.length);
    int dataIndex = 0;
    int dataLength = data.length * 8;

    for (int i = 0; i < audio.length; i++) {
        if (dataIndex < dataLength) {
            int bit = (data[dataIndex / 8] >> (7 - (dataIndex % 8))) & 1;
            result[i] = (byte)((audio[i] & 0xFE) | bit);
            dataIndex++;
        }
    }
    return result;
}

public static byte[] extractData(byte[] audio, int dataSize) {
    byte[] data = new byte[dataSize];
    int dataIndex = 0;

    for (int i = 0; i < audio.length; i++) {
        if (dataIndex < dataSize * 8) {
            int bit = audio[i] & 1;
            data[dataIndex / 8] = (byte)((data[dataIndex / 8] << 1) | bit);
            dataIndex++;
        }
    }
    return data;
}

```

```

    }
}

// GUI implementation
public static class SteganographyGUI extends JFrame {
    private JTextField messageField;
    private JLabel imageFileLabel;
    private JLabel audioFileLabel;
    private File imageFile;
    private File audioFile;
    private JTextArea resultArea;

    public SteganographyGUI() {
        setTitle("Hybrid Model Comparison");
        setSize(600, 400);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(new BorderLayout());

        // Panel for inputs
        JPanel inputPanel = new JPanel();
        inputPanel.setLayout(new GridLayout(5, 2));

        // Message input
        inputPanel.add(new JLabel("Message:"));
        messageField = new JTextField();
        inputPanel.add(messageField);

        // Image file selection
        inputPanel.add(new JLabel("Cover Image:"));
        imageFileLabel = new JLabel("No file selected");
        JButton imageButton = new JButton("Select Image");

```

```

imageButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        JFileChooser fileChooser = new JFileChooser();
        fileChooser.setFileFilter(new FileNameExtensionFilter("Image files", "png", "jpg",
"bmp"));
        int returnValue = fileChooser.showOpenDialog(null);
        if (returnValue == JFileChooser.APPROVE_OPTION) {
            imageFile = fileChooser.getSelectedFile();
            imageFileLabel.setText(imageFile.getName());
        }
    }
});
inputPanel.add(imageFileLabel);
inputPanel.add(imageButton);

// Audio file selection
inputPanel.add(new JLabel("Cover Audio:"));
audioFileLabel = new JLabel("No file selected");
JButton audioButton = new JButton("Select Audio");
audioButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        JFileChooser fileChooser = new JFileChooser();
        fileChooser.setFileFilter(new FileNameExtensionFilter("Audio files", "wav"));
        int returnValue = fileChooser.showOpenDialog(null);
        if (returnValue == JFileChooser.APPROVE_OPTION) {
            audioFile = fileChooser.getSelectedFile();
            audioFileLabel.setText(audioFile.getName());
        }
    }
});

```

```

});
inputPanel.add(audioFileLabel);
inputPanel.add(audioButton);

// Result area
resultArea = new JTextArea();
resultArea.setEditable(false);
JScrollPane scrollPane = new JScrollPane(resultArea);

// Buttons
JButton embedButton = new JButton("Embed and Extract");
embedButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        try {
            performSteganography();
        } catch (Exception ex) {
            ex.printStackTrace();
            JOptionPane.showMessageDialog(null, "An error occurred: " + ex.getMessage(),
"Error", JOptionPane.ERROR_MESSAGE);
        }
    }
});

// Add components to frame
add(inputPanel, BorderLayout.NORTH);
add(scrollPane, BorderLayout.CENTER);
add(embedButton, BorderLayout.SOUTH);
}

private void performSteganography() throws Exception {

```



```

        if (imageFile == null || audioFile == null) {
            JOptionPane.showMessageDialog(null, "Please select both an image and an audio
file.", "Error", JOptionPane.ERROR_MESSAGE);
            return;
        }

        String message = messageField.getText();
        if (message.isEmpty()) {
            JOptionPane.showMessageDialog(null, "Please enter a message.", "Error",
JOptionPane.ERROR_MESSAGE);
            return;
        }

        // Generate encryption key
        SecretKey key = EncryptionUtil.generateKey();

        // Encrypt data
        byte[] encryptedData = EncryptionUtil.encrypt(message.getBytes(), key);

        // Image steganography
        BufferedImage image = ImageIO.read(imageFile);
        if (image == null) {
            JOptionPane.showMessageDialog(null, "Error reading cover image.", "Error",
JOptionPane.ERROR_MESSAGE);
            return;
        }

        // Check image capacity
        int imageCapacity = image.getWidth() * image.getHeight();
        if (encryptedData.length * 8 > imageCapacity) {

```

```

        JOptionPane.showMessageDialog(null, "Data too large to embed in the image.",
"Error", JOptionPane.ERROR_MESSAGE);
        return;
    }

    // File size before embedding
    long imageSizeBefore = imageFile.length();

    // Time tracking for image steganography
    long startTimeImage = System.nanoTime();

    BufferedImage stegoImage = ImageSteganography.embedData(image, encryptedData);
    ImageIO.write(stegoImage, "png", new File("stego_image.png"));

    long endTimeImage = System.nanoTime();
    long timeTakenImage = endTimeImage - startTimeImage;

    // File size after embedding
    File stegoImageFile = new File("stego_image.png");
    long imageSizeAfter = stegoImageFile.length();

    // Extract and decrypt the message
    BufferedImage extractedImage = ImageIO.read(stegoImageFile);
    byte[] extractedDataImage = ImageSteganography.extractData(extractedImage,
encryptedData.length);
    byte[] decryptedDataImage = EncryptionUtil.decrypt(extractedDataImage, key);
    String imageResult = "Decrypted message from image: " + new
String(decryptedDataImage) + "\n"
        + "Time taken (image): " + timeTakenImage / 1_000_000.0 + " ms\n"
        + "Image size before: " + imageSizeBefore / 1024 + " KB, after: " + imageSizeAfter
/ 1024 + " KB";

```

```

// Audio steganography
byte[] audio = Files.readAllBytes(audioFile.toPath());

// Check audio capacity
int audioCapacity = audio.length;
if (encryptedData.length * 8 > audioCapacity) {
    JOptionPane.showMessageDialog(null, "Data too large to embed in the audio.",
    "Error", JOptionPane.ERROR_MESSAGE);
    return;
}

// File size before embedding
long audioSizeBefore = audioFile.length();

// Time tracking for audio steganography
long startTimeAudio = System.nanoTime();

byte[] stegoAudio = AudioSteganography.embedData(audio, encryptedData);
Files.write(Paths.get("stego_audio.wav"), stegoAudio);

long endTimeAudio = System.nanoTime();
long timeTakenAudio = endTimeAudio - startTimeAudio;

// File size after embedding
File stegoAudioFile = new File("stego_audio.wav");
long audioSizeAfter = stegoAudioFile.length();

// Extract and decrypt the message
byte[] extractedAudio = Files.readAllBytes(stegoAudioFile.toPath());

```

```

        byte[] extractedDataAudio = AudioSteganography.extractData(extractedAudio,
encryptedData.length);
        byte[] decryptedDataAudio = EncryptionUtil.decrypt(extractedDataAudio, key);
        String audioResult = "Decrypted message from audio: " + new
String(decryptedDataAudio) + "\n"
            + "Time taken (audio): " + timeTakenAudio / 1_000_000.0 + " ms\n"
            + "Audio size before: " + audioSizeBefore / 1024 + " KB, after: " + audioSizeAfter /
1024 + " KB";

        // Display results
        resultArea.setText(imageResult + "\n\n" + audioResult);
    }
}

public static void main(String[] args) {
    SwingUtilities.invokeLater(new Runnable() {
        @Override
        public void run() {
            new SteganographyGUI().setVisible(true);
        }
    });
}
}

```