



CHRIST
(DEEMED TO BE UNIVERSITY)
BANGALORE • INDIA

A JAVA PROJECT ON

Fortifying Data Security through AES Encryption and Decryption: CryptiFY

Asmita Mondal [2348018]

PROJECT GUIDE: Dr. Vyshali Gogi

Submitted to the Department of Statistics and Data Science in partial fulfilment of the
requirements for the degree of M.Sc. Data Science

CERTIFICATE OF AUTHENTICATED WORK

This is to certify that the project report entitled “Fortifying Data Security through AES Encryption and Decryption: CryptiFY” submitted to Department of Statistics and Data Science , Christ (Deemed to be University), Bangalore, Central Campus in partial fulfilment of the requirement for the award of the degree of Master of Science (M. Sc.) is an original work carried out by:

Name	Register No.	Class
ASMITA MONDAL	2348018	3MDS

under the guidance of Dr. Vyshali Gogi. The matter embodied in this project is authentic and is genuine work done by the student and has not been submitted whether to this College or to any other Institute for the fulfilment of the requirement of any course of study.

DECLARATION: I affirm that I have identified all my sources and that no part of my dissertation paper uses unacknowledged materials.



Signature of the Guide

Signature of the Student

Dr. Vyshali Gogi

.....

Name of the Guide

14-04-2024

.....

Date

ACKNOWLEDGEMENT

I would like to sincerely thank Dr. Vyshali Gogi for her invaluable advice and assistance with my project. Her extensive knowledge and skill in the fields of data science and Java have been crucial to the development and success of my project. I am appreciative of the time and effort she has spent on me. I would also like to express my profound gratitude to my peers who have given their time and energy to my project by commenting on my work, taking part in discussions, or contributing their thoughts.

I would also like to thank our Head of Department Dr. Saleema, for giving us the tools and resources we needed to finish our project. I appreciate her never-ending encouragement and support, as well as the environment she helped to foster for creativity and innovation. Additionally, I extend my heartfelt thanks to my parents and supportive friends for their unwavering encouragement and belief in my abilities.

ABSTRACT

The project "CryptiFY" aims to develop a Java-based application for secure file encryption and decryption, with a focus on data security in ethical data science practices. Users can encrypt files of various formats using the AES encryption technique, facilitating secure data handling, particularly for large files. The application incorporates a two-step authentication process, where encrypted files, secret keys, Initialization Vectors (IVs), and user-defined secret codes are stored in a database. The secret code is authenticated and protected by robust checking and a combination of Vigenère and Caesar Ciphers. Decryption is only permitted when the provided code matches the stored combination. This project works on files of type text, word document, csv files and excel files. Thus, the project contributes to enhancing data security protocols in the field of data science, ensuring the confidentiality and integrity of sensitive information. Notable software used includes Java (version 21.0.1 LTS), JDBC (version 8.3.0 of the MySQL Connector/J JDBC driver), MySQL Workbench (version 8.0), and Eclipse IDE (version 2024-03).

TABLE OF CONTENTS

ACKNOWLEDGEMENT.....	3
ABSTRACT.....	4
LIST OF FIGURES.....	6
CHAPTER 1: INTRODUCTION.....	8
What is Encryption and Decryption?.....	8
Objective of the Project.....	9
Encryption End.....	9
Decryption End.....	9
Purpose of the Project.....	9
Features of the Project.....	10
Applicability of the Project.....	10
CHAPTER 2: LITERATURE REVIEW.....	11
CHAPTER 3: SURVEY OF TECHNOLOGIES.....	14
Java.....	14
JavaFX.....	14
MySQL.....	15
Classical Substitution Ciphers.....	15
CHAPTER 4: REQUIREMENTS AND SPECIFICATIONS.....	18
Problem Definition.....	18
Planning the Flow.....	18
Requirement Specifications.....	18
Used For This Project:.....	18
Software:.....	18
Hardware:.....	19
General Requirements:.....	20
Software.....	20
Hardware.....	21

CHAPTER 5: PROPOSED METHODOLOGY.....	22
Flow Diagram.....	22
Encryption Process:.....	22
Decryption Process:.....	22
Explanation.....	23
CHAPTER 6: DESIGN AND IMPLEMENTATION.....	24
Encryption Code.....	24
Decryption Code.....	31
CSS Styling.....	36
Database and Table Creation.....	37
module-info.java.....	37
CHAPTER 8: RESULTS AND DISCUSSIONS.....	38
Running the Project.....	38
Observations.....	41
CHAPTER 9: CONCLUSION.....	42
Limitations.....	42
Future Scope.....	42
REFERENCES.....	43

LIST OF FIGURES AND TABLES

1. Figure 1: AES Encryption First Round	17
2. Figure 2: Encryption FlowChart	22
3. Figure 3: Decryption FlowChart	22
4. Figure 4: Interface and Success Alert Box	38
5. Figure 5: Table in Database	38
6. Figure 6: Decryption Interface and Success Message	39
7. Figure 7: Error Display with wrong Key or IV Input	39
8. Figure 8: Encrypted vs Decrypted CSV Text Example	40
9. Table 1: Comparison of Time for Different File Types	41

CHAPTER 1: INTRODUCTION

In contemporary society, data security has emerged as a critical concern due to the escalating frequency and severity of data breaches. These breaches represent instances where sensitive information is compromised, leading to severe consequences for individuals, organizations, and even nations. For example, the Equifax breach exposed Social Security numbers, birth dates, and addresses of approximately 147 million individuals. Similarly, Yahoo's breach affected over 3 billion user accounts. These incidents highlight the crucial role of encryption in safeguarding data against unauthorized access and exploitation.

Had the data in these breaches been properly encrypted, even if accessed by unauthorized parties, it would have been unintelligible and thus significantly less valuable. Encryption serves as a crucial defense mechanism against data breaches, ensuring that even if attackers gain access to data, they cannot decipher its contents without the appropriate decryption keys. Therefore, the development of robust encryption solutions, such as the proposed "CryptiFY" application, is imperative in addressing the persistent threat of data breaches and enhancing overall data security in the digital age.

What is Encryption and Decryption?

Encryption is the process of converting plain text or readable data into an unintelligible form known as ciphertext, using cryptographic algorithms and keys. This transformation ensures that the original data is protected from unauthorized access or interception, as only individuals with the corresponding decryption key can revert the ciphertext back to its original form through decryption. Encryption plays a critical role in safeguarding sensitive information, such as personal data, financial transactions, and confidential communications, from potential security threats and breaches by ensuring its confidentiality and integrity.

Decryption, on the other hand, is the reverse process of encryption, where ciphertext is transformed back into plaintext using the appropriate decryption algorithm and key. Authorized parties possessing the decryption key can decrypt the ciphertext, thereby accessing and interpreting the original data. This process of encryption and decryption serves as the foundation for secure communication, data storage, and authentication mechanisms, providing a robust framework for protecting digital assets and ensuring compliance with regulatory standards and data protection laws.

Objective of the Project

The project aims to achieve the following objectives:

Encryption End

- Accept user input for the secret code, file to be encrypted, and desired key size.
- Encrypt the file using AES encryption and the secret code with a combination of Vigenère and Caesar ciphers.
- Store the secret key, IV, encrypted secret code, and encrypted file in a database.

Decryption End

- Accept encrypted file, secret code, secret key, and IV from the user.
- Retrieve the encrypted secret code from the database based on the secret key and IV combination.
- Decrypt the encrypted secret code with the given user input secret code.
- Decrypt the file if the decrypted secret code, secret key, and IV combination match the database records.

Purpose of the Project

The purpose of this project is to address the critical need for robust data security measures in an increasingly digital world. With the proliferation of sensitive information stored and transmitted electronically, the risk of unauthorized access, data breaches, and cyberattacks has become ever-present. By focusing on encryption and decryption techniques, the project aims to fortify data security and protect valuable information from potential threats. Through the implementation of advanced encryption algorithms such as AES (Advanced Encryption Standard) in Java, the project seeks to establish a reliable framework for safeguarding sensitive data across various platforms and environments.

Furthermore, the project aspires to empower individuals and organizations with the tools and knowledge necessary to mitigate risks and enhance data privacy and confidentiality. By promoting awareness of encryption best practices and providing accessible solutions for implementing encryption mechanisms, the project endeavors to foster a culture of data security and resilience against evolving cybersecurity challenges. Ultimately, the project's overarching goal is to contribute to the creation of a safer digital landscape where individuals can confidently engage in online activities, businesses can protect their assets and reputation, and society as a whole can benefit from the secure exchange of information.

Features of the Project

- 01. Secure File Transmission:** Encrypt files before transmission, ensuring that only authorized users with the secret key, code, and IV can decrypt them.
- 02. Confidential Data Storage:** Safely store sensitive files in a database, protecting them from unauthorized access. Only the encrypted file and encrypted code rather than the originals are stored in the database ensuring added security in the storage end.
- 03. Data Integrity Verification:** Verify the integrity of encrypted files by comparing cryptographic hashes before and after transmission.
- 04. Layered Authentication:** The application uses AES encryption along with a combination of Vigenere and Caesar techniques for encryption and decryption.
- 05. Supported Files:** The project works on **.csv**, **.txt**, **.docx**, **.doc** and **.xlsx** files.

Applicability of the Project

- 01. Healthcare Sector:** In the healthcare industry, patient records contain sensitive information such as medical history, diagnoses, and treatment plans. Implementing "CryptiFY" can ensure the secure encryption of these records, safeguarding patient confidentiality.
- 02. Financial Institutions:** Financial institutions deal with highly confidential data such as customer financial records, transaction details, and account information. "CryptiFY" can be used to encrypt these sensitive data files, protecting them from unauthorized access.
- 03. Legal Firms:** Legal firms handle sensitive legal documents, contracts, and case files that require confidentiality and integrity. By using "CryptiFY," legal professionals can encrypt these documents securely.
- 04. Research Organizations:** Research organizations often deal with proprietary data, intellectual property, and sensitive research findings. "CryptiFY" can help researchers encrypt their data files securely, protecting their intellectual property and research integrity from unauthorized access or theft.

CHAPTER 2: LITERATURE REVIEW

A year-wise survey of relevant works is listed below:

2009

1. Abdul et al. [6] conducted an extensive comparison of six prevalent encryption algorithms, including AES (Rijndael), DES, 3DES, RC2, BLOWFISH, and RC6. Their study evaluated the performance of these algorithms across various settings, including different data block sizes, data types, battery power consumption, key sizes, and encryption/decryption speeds. The results revealed that there was no significant difference between hexadecimal base encoding and base 64 encoding. When considering packet size variations, BLOWFISH exhibited superior performance compared to other common encryption algorithms, with RC6 following closely. However, when the data type was changed to images instead of text, RC2, RC6, and BLOWFISH demonstrated a time consumption disadvantage. Additionally, 3DES exhibited lower performance compared to the DES algorithm. Finally, altering the key size significantly impacted battery and time consumption, indicating a clear correlation between higher key sizes and resource utilization.

2014

1. Kundankumar Rameshwar Saraf et al. [7] emphasized the increasing importance of information security in the era of rapid digital data exchange, particularly in contexts such as internet banking and email account management. They highlighted the need for robust protection mechanisms for both text and image data, with the National Institute of Standards and Technology (NIST) spearheading the development of the Advanced Encryption Standard (AES) as a replacement for the Data Encryption Standard (DES). Recognizing the diverse nature of data types and their unique security requirements, the authors implemented AES encryption and decryption for both text and image data. For text encryption, 128-bit inputs were synthesized and simulated on a TMS320C6713 DSP processor using Code Composer Studio tool. Meanwhile, image encryption was carried out using Java code synthesized and simulated by the Java Application Platform SDK, with a focus on Code Block Chaining (CBC) mode with PKCS 5 padding. This project showcases the versatility of AES encryption clearly concluding that AES was faster than all other algorithms.

2015

1. Abraham Lemma et al. [5] explored the expanding landscape of e-commerce. The complexity of security requirements, encompassing aspects like confidentiality and authentication, underscores the challenge. Focusing on DES, TripleDES, AES, and Blowfish, the authors conducted a comparative analysis, evaluating their performance in terms of encryption and decryption time, throughput, and memory usage. From the results presented in this paper, it was concluded that TripleDES required more time for encryption/decryption, utilized less memory, and exhibited lower throughput. Both AES and Blowfish demonstrated comparable encryption/decryption times and superior throughput, with AES requiring more memory than Blowfish. Additionally, DES consumed similar memory to TripleDES but achieved minimal encryption/decryption time and higher throughput compared to TripleDES.
2. Tayde and Siledar [2] examined the rise of smart gadgets like smartphones and tablets, noting their portability and extensive functionalities akin to traditional computers. Smartphones have encountered security challenges, prompting the exploration of encryption as a solution for safeguarding data during storage and transmission. The study reviewed various encryption algorithms, including DES, 3DES, Blowfish, and RSA, highlighting their limitations such as small key sizes and slower processing. To address these concerns, the authors advocated for the implementation of the Advanced Encryption Standard (AES) algorithm, renowned for its robust security and high speed. The application showcased AES's efficacy in encrypting and decrypting various file types, from text to images, on the Android platform, thereby contributing to enhanced data security in activities such as banking, production, and research.

2016

1. Amola et al. [3] identified security as a critical aspect in software development, particularly concerning the handling of sensitive data. They recognized SQLite, commonly used in Android applications, as a lightweight yet inherently insecure database lacking encryption capabilities. Acknowledging the vulnerability of data stored in SQLite, especially in the event of device compromise, they embarked on a project aimed at bolstering data security through encryption. Leveraging the Advanced Encryption Standard (AES), a renowned symmetric encryption method, the team implemented AES in Java to fortify data stored in the SQLite database. They drew inspiration from the Android Open Source Project (AOSP), where the inadequacy of SQLite security was highlighted. The project's core objective centered on demonstrating AES's efficacy in securing SQLite databases within Android applications.

2021

1. Zhimao Lu et al. [1] presented a comprehensive analysis of data encryption technology in the context of modern internet and e-commerce trends. They highlighted the pivotal role of encryption algorithms, particularly the widely used Advanced Encryption Standard (AES), in ensuring robust data security. While AES was lauded for its effectiveness in symmetric encryption, challenges arose in key management and security functions. The authors meticulously examined these issues, proposing optimizations such as round key reduction and enhanced key scheduling, alongside an integration with the RSA algorithm. Leveraging Java's rich libraries, their methodology evaluated the proposed enhancements through a rigorous comparison of encryption/decryption speed, entropies, and memory consumption against traditional algorithms. The results showcased the superior performance and heightened security of the hybrid AES-RSA approach, particularly in secure file sharing over vulnerable channels. This study offered valuable insights for businesses seeking tailored encryption solutions to address diverse security needs.

2023

1. Saleha Saudagar et al. [4] underscored the intrinsic value of data across industries, particularly accentuated in the digital era with heightened data exchange via text and media files. While various methods like VPNs, FTP, and SMTP have been employed to secure data exchange pathways, the susceptibility of actual data remains a concern. To address this, the paper proposed a Java-based system leveraging encryption algorithms to independently secure data apart from the established pathways. The system encrypted and decrypted media files using three distinct algorithms tailored for diverse file types including mp4, png, jpg, pdf, doc, xml, etc. By focusing on data encryption, the study aims to fortify data security and mitigate vulnerabilities associated with data exchange protocols.

CHAPTER 3: SURVEY OF TECHNOLOGIES

Java

Java, a robust and versatile programming language, has solidified its position as a fundamental technology in software development. Offering a platform-independent environment through its "write once, run anywhere" principle, Java allows developers to create applications that can seamlessly execute on various platforms without requiring platform-specific modifications. This portability is facilitated by the Java Virtual Machine (JVM), which interprets Java bytecode and translates it into machine code at runtime. Furthermore, Java's object-oriented nature promotes code reusability, maintainability, and scalability, enabling developers to build complex systems with ease. Additionally, Java's extensive standard library provides a wide range of pre-built classes and utilities for common tasks, reducing development time and effort.

The Java ecosystem extends beyond the core language to include a plethora of libraries, frameworks, and tools that augment development capabilities. From enterprise-level frameworks like Spring and Hibernate to web development frameworks like Spring Boot and Vaadin, Java offers solutions for various application domains. Moreover, the vibrant Java community actively contributes to the language's evolution and support, providing resources, tutorials, and forums for developers to collaborate and learn. With its enduring popularity and continuous innovation, Java remains a stalwart choice for building robust, scalable, and cross-platform applications across diverse industries and use cases.

JavaFX

JavaFX, a powerful framework for building rich client applications, empowers developers to create visually appealing and interactive user interfaces (UIs) with ease. Leveraging Java's strengths in portability and performance, JavaFX provides a modern toolkit for designing UIs that are not only aesthetically pleasing but also responsive and intuitive. Its declarative markup language, FXML, allows developers to separate UI design from application logic, facilitating collaboration between designers and developers. Furthermore, JavaFX's scene graph-based architecture enables developers to create complex UI layouts and visualizations using a hierarchical structure of nodes, enhancing flexibility and scalability. Additionally, JavaFX's support for CSS styling enables developers to customize the appearance of UI components, achieving consistent branding and visual coherence across applications.

Beyond its UI capabilities, JavaFX offers robust multimedia and graphics support, making it suitable for developing applications with advanced multimedia features. From embedding videos and audio to creating sophisticated animations and 3D graphics, JavaFX provides tools and APIs for incorporating rich media content into applications. Moreover, JavaFX's event-driven architecture simplifies handling user interactions and input, allowing developers to create dynamic and responsive user experiences. With its seamless integration with Java libraries and frameworks, JavaFX empowers developers to build cross-platform GUI applications that deliver immersive and engaging user experiences across desktop, mobile, and embedded devices.

MySQL

MySQL, a widely-used open-source relational database management system (RDBMS), has become a cornerstone technology for storing, managing, and retrieving structured data. Renowned for its scalability, reliability, and performance, MySQL caters to a diverse range of applications spanning from small-scale websites to large enterprise systems. Its robust transaction support, ACID compliance, and support for various storage engines make it a versatile choice for building data-driven applications. Additionally, MySQL's extensive support for SQL queries and data manipulation operations enables developers to efficiently interact with the database and retrieve information as needed. Moreover, MySQL's compatibility with different programming languages and platforms makes it a preferred choice for developers seeking a powerful and cost-effective database solution.

Classical Substitution Ciphers

Caesar ciphers, named after Julius Caesar, are one of the simplest and oldest encryption techniques. They operate by shifting each letter of the plaintext by a fixed number of positions in the alphabet. For example, with a shift of 3, 'A' would become 'D', 'B' would become 'E', and so forth. This shift, known as the cipher key or shift key, remains constant throughout the encryption process. While straightforward to implement, Caesar ciphers are easily breakable through brute-force methods due to their limited key space, as there are only 25 possible shifts in the English alphabet.

Vigenère ciphers, on the other hand, introduce a level of complexity by using a keyword to determine the amount of shift applied to each letter. This keyword is repeated cyclically to match the length of the plaintext. For example, if the keyword is 'KEY', and the plaintext is 'HELLO', the keyword would be repeated as 'KEYKE' to match the length of the plaintext. Each letter of the plaintext is then shifted by the corresponding letter of the keyword. Unlike Caesar ciphers, where each letter is shifted by the same amount, Vigenère ciphers employ a variable shift based on the letters of the keyword. While more secure than Caesar ciphers, Vigenère ciphers can still be susceptible to cryptanalysis if the keyword is short or easily guessable. Despite their vulnerabilities, both Caesar and Vigenère ciphers serve as foundational examples of classical encryption techniques and provide insights into the evolution of cryptographic methods.

AES Encryption

AES (Advanced Encryption Standard), a symmetric encryption algorithm, plays a pivotal role in ensuring data security and confidentiality in modern cryptographic systems. Adopted by governments, financial institutions, and industries worldwide, AES offers a high level of security and efficiency, making it suitable for a wide range of applications. With its block cipher design and variable key lengths (128-bit, 192-bit, and 256-bit), AES provides robust protection against cryptographic attacks while maintaining computational efficiency. Moreover, AES's simplicity and widespread adoption make it a preferred choice for securing sensitive data during transmission and storage.

Process Flow

AES Encryption:

1. **Key Expansion:** Generate a set of round keys from the original encryption key using the Key Expansion algorithm. This process involves performing a series of transformations on the key to create a unique key for each round of encryption.
2. **Initial Round:** Add the initial round key to the plaintext block. The initial round key is derived from the encryption key.
3. **Rounds:** Perform a specified number of encryption rounds, each consisting of several transformation steps.
4. **SubBytes:** Substitute each byte of the state matrix with a corresponding byte from the S-box lookup table.
5. **ShiftRows:** Shift the rows of the state matrix cyclically to the left by different offsets.
6. **MixColumns:** Combine the bytes of each column in the state matrix using a fixed polynomial matrix multiplication.
7. **AddRoundKey:** XOR the state matrix with the round key derived from the key schedule.
8. **Final Round:** Perform a final encryption round without the MixColumns step.
9. **Output:** The resulting state matrix after the final round represents the ciphertext.

AES Decryption:

1. **Key Expansion:** Generate the round keys using the same Key Expansion algorithm as in encryption.
2. **Initial Round:** Add the initial round key to the ciphertext block.
3. **Rounds:** Perform a specified number of decryption rounds, each consisting of several transformation steps in reverse order compared to encryption:
4. **InverseShiftRows:** Reverse the row shifts performed during encryption.
5. **InverseSubBytes:** Substitute each byte of the state matrix with a corresponding byte from the inverse S-box lookup table.
6. **AddRoundKey:** XOR the state matrix with the round key derived from the key schedule.
7. **InverseMixColumns:** Combine the bytes of each column in the state matrix using a fixed polynomial matrix multiplication in reverse.
8. **Final Round:** Perform a final decryption round without the InverseMixColumns step.
9. **Output:** The resulting state matrix after the final round represents the plaintext.

AES Encryption Algorithm in Java JCE Library

The Java Cryptography Extension (JCE) library provides a comprehensive set of cryptographic services and algorithms for securing data in Java applications. Within the JCE library, the AES encryption algorithm is implemented using various modes of operation, including Electronic Codebook (ECB), Cipher Block Chaining (CBC), Cipher Feedback (CFB), and Output Feedback (OFB). Developers can leverage the Cipher class and its subclasses to perform AES encryption and decryption operations seamlessly.

By specifying the desired encryption mode, padding scheme, and key size, developers can customize AES encryption to suit their specific security requirements. Additionally, the JCE library offers support for key generation, key management, and secure random number generation, enhancing the overall security of AES-encrypted data. With its robust implementation and integration capabilities, the AES encryption algorithm in the Java JCE library enables developers to build secure and resilient Java applications that protect sensitive information from unauthorized access and tampering.

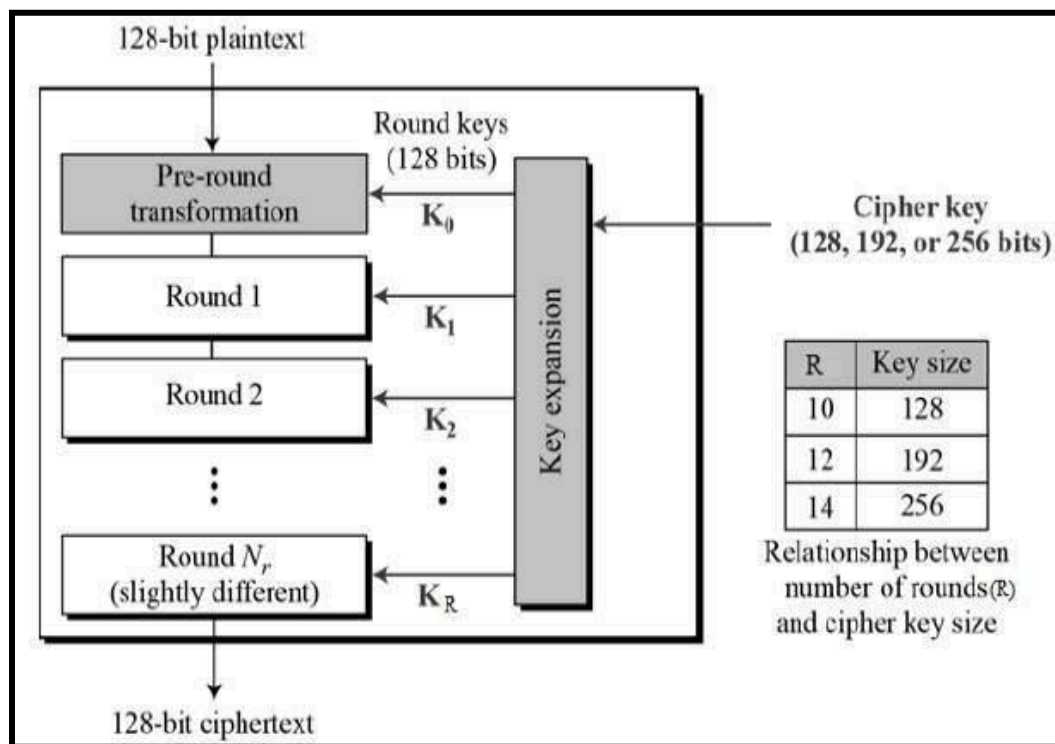


Figure 1: AES Encryption First Round

CHAPTER 4: REQUIREMENTS AND SPECIFICATIONS

Problem Definition

To develop an application that ensures smooth encryption and decryption of files using Java. Given the significance of data security, there is a need for a robust solution that can effectively encrypt and decrypt files, mitigating the risk of unauthorized access and data breaches.

Planning the Flow

- To create a concise problem statement and conduct background research on similar work.
- To learn about Java libraries required for AES encryption.
- To understand JavaFX dependencies and project creation.
- To configure the environment in Eclipse IDE and setup Database dependencies using JDBC Driver Manager with MySQL.
- To code and test run on documents.
- To create UI for better interaction.
- To document, report, and present the project.

Requirement Specifications

Used For This Project:

Software:

Development Environment:

- Eclipse IDE for Java Developers (Version 2024-03)
- MySQL Workbench (Version 8.0)
- Java Development Kit (JDK) (Version 21.0.1 LTS)

Java Libraries and Dependencies:

- Java Cryptography Extension (JCE)
- JavaFX SDK (Version 22)
- Connector/J JDBC Driver (Version 8.3.0)

*Hardware:***Computer System:**

- Dell Inspiron 15 3525 Laptop

Processor:

- AMD Ryzen 7 5825U with AMD Radeon (TM) Graphics, 2000 Mhz, 8 Core(s), 16 Logical Processor(s)

Memory (RAM):

- 16GB RAM

Operating System:

- Microsoft Windows 11 Home

Display:

- Display Size: 15.6 inches
- Display Resolution: Full HD WVA Display

General Requirements:

Software

Integrated Development Environment (IDE):

- Any Java-compatible IDE such as Eclipse, IntelliJ IDEA, or NetBeans.

Database Management Software:

- MySQL Database Server, PostGRES or any other compatible relational database management system (RDBMS).
- MySQL Workbench or any other database management tool for database administration and schema design.

Java Development Kit (JDK):

- Latest version of Java Development Kit (JDK), preferably LTS version.
- Ensure compatibility with the chosen IDE and operating system.

Java Cryptography Library:

- Java Cryptography Extension (JCE) for standard cryptographic functionalities.
- Alternatively, third-party libraries like Bouncy Castle for extended cryptographic capabilities.

JavaFX SDK:

- Latest version of JavaFX Software Development Kit (SDK) for building JavaFX applications.

MySQL Connector/J JDBC Driver:

- MySQL Connector/J JDBC driver for Java database connectivity with MySQL database server.

Hardware

Computer System:

- Desktop or Laptop Computer capable of running the chosen IDE and JDK.
- Adequate processing power and memory for development tasks.

Memory (RAM):

- At least 4GB RAM recommended, higher RAM for better performance.

Storage:

- Adequate Storage Space for software installations, project files, and database storage.

Operating System:

- Any compatible operating system supported by the chosen IDE and JDK, preferably Windows 10 and above.

CHAPTER 5: PROPOSED METHODOLOGY

Flow Diagram

Encryption Process:

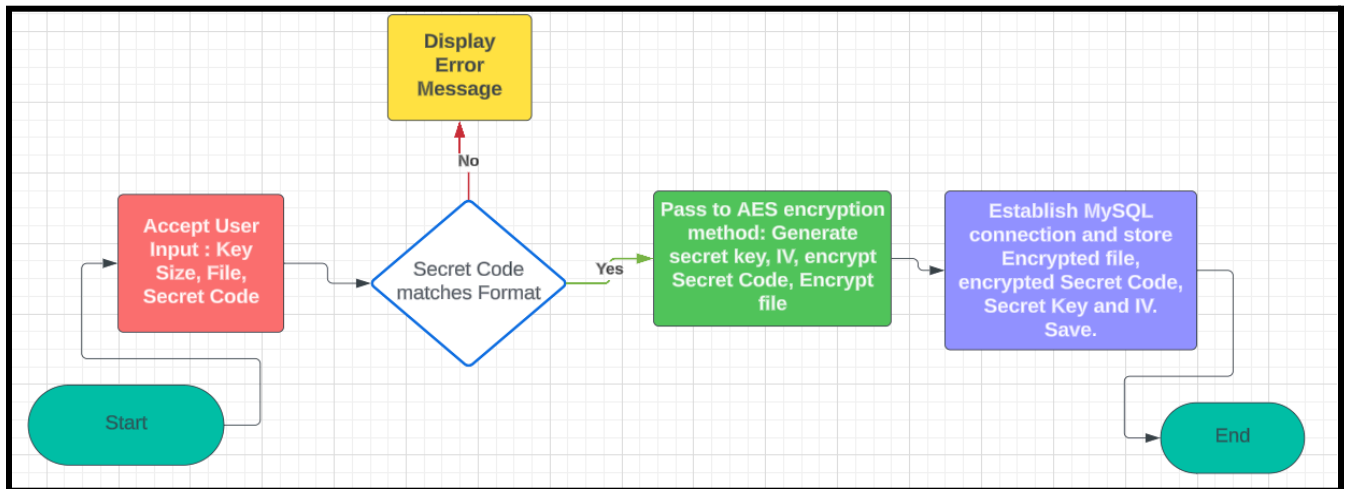


Figure 2: Encryption FlowChart

Decryption Process:

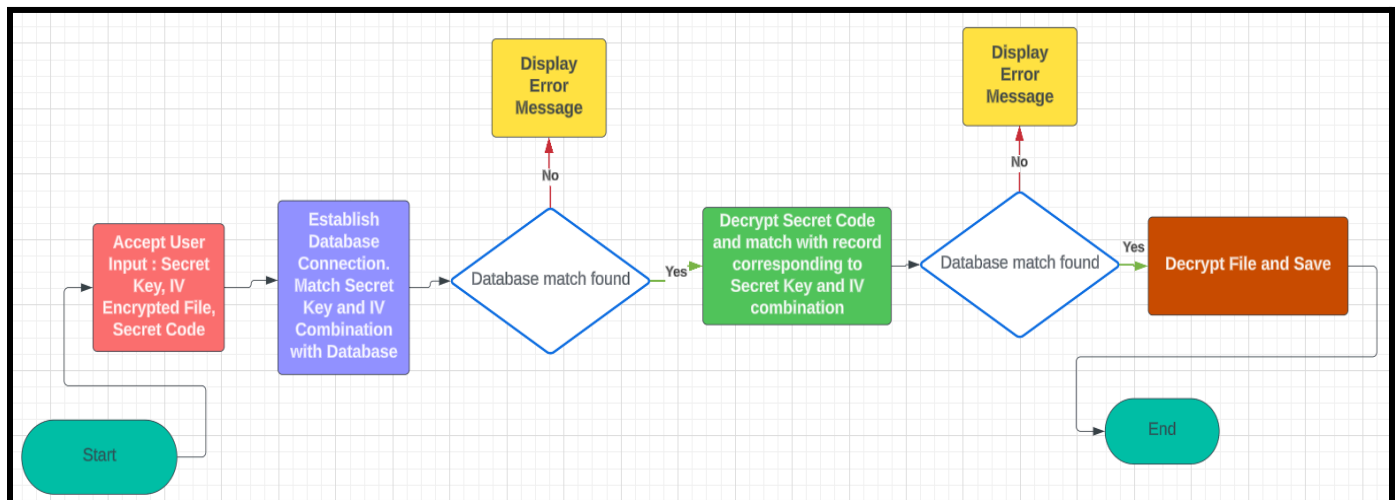


Figure 3: Decryption FlowChart

Explanation

- **Learning Java Libraries for AES Encryption:** Research and identify a suitable Java cryptographic library for AES encryption, such as Java Cryptography Extension (JCE). Study the documentation and examples provided by the chosen library, understanding its usage and capabilities in the context of JavaFX applications. Experiment with sample code to gain proficiency in implementing AES encryption and decryption algorithms.
- **Understanding JavaFX Dependencies and Project Creation:** Familiarize with JavaFX, including its architecture, components, and dependencies. Install the necessary JavaFX libraries and SDK for the development environment. Create a new JavaFX project in Eclipse IDE, configuring it to include JavaFX libraries and dependencies. Utilize CSS for styling the JavaFX user interface to enhance the visual appeal and user experience of the application.
- **Configuring Environment in Eclipse IDE and Setting up Database Dependencies:** Install and configure Eclipse IDE for Java development, ensuring it is set up to work with JavaFX and external libraries. Integrate MySQL with Eclipse by adding the Connector/J JDBC driver to the project's build path. Establish a connection to the MySQL database using JDBC DriverManager, configuring the connection parameters appropriately. Utilize MySQL Workbench for database management and schema design, ensuring seamless integration with the JavaFX application.
- **Coding and Testing AES Encryption on Documents:** Implement AES encryption and decryption algorithms within JavaFX application, utilizing the chosen cryptographic library. Generate encryption keys using the cryptographic library, ensuring proper initialization vector (IV) specification (e.g., IvParameterSpec). Develop functionality to encrypt and decrypt documents. Write comprehensive unit tests to verify the correctness and robustness of the encryption and decryption processes. Create user defined functions for Vigenère and Caesar Ciphers to encrypt and decrypt Secret Code.
- **Creating UI for Better Interaction:** Design and implement a user-friendly graphical user interface (GUI) using JavaFX controls and layouts, incorporating CSS for styling.

CHAPTER 6: DESIGN AND IMPLEMENTATION

Encryption Code

```
package application;

import javafx.application.Application;
import javafx.geometry.Insets;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.layout.GridPane;
import javafx.stage.FileChooser;
import javafx.stage.Stage;

import javax.crypto.*;
import javax.crypto.spec.IvParameterSpec;
import java.io.*;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.util.Base64;

public class Main extends Application {

    private TextField secretCodeField;
    private TextField filePathField;
    private ComboBox<Integer> keySizeComboBox;

    @Override
    public void start(Stage primaryStage) throws Exception {

        primaryStage.setTitle("Asmita Roy Mondal");

        // Create header label
        Label headerLabel = new Label("CRYPTIFY ENCRYPTION");
        headerLabel.setStyle("-fx-font-size: 24px; -fx-font-weight: bold; -fx-background-color: #7B68EE; -fx-padding: 10px; -fx-text-fill: white; -fx-background-radius: 5px;");
        // Create UI components
        Label secretCodeLabel = new Label("Secret Code:");
        // Replace the existing TextField with PasswordField
        PasswordField secretCodeField = new PasswordField();
        secretCodeField.setPromptText("Enter Secret Code");
        this.secretCodeField = secretCodeField;
```



```

Label filePathLabel = new Label("File Path:");
filePathField = new TextField();
filePathField.setPromptText("Select File");

Button chooseFileButton = new Button("Choose File");
chooseFileButton.setOnAction(e -> chooseFile(primaryStage));

Label keySizeLabel = new Label("Key Size:");
keySizeComboBox = new ComboBox<>();
keySizeComboBox.getItems().addAll(128, 192, 256);
keySizeComboBox.setValue(128);

Button encryptButton = new Button("Encrypt");
encryptButton.setOnAction(e -> encryptFile());

// Create layout
GridPane grid = new GridPane();
grid.setPadding(new Insets(20));
grid.setVgap(10);
grid.setHgap(10);

grid.add(headerLabel, 0, 0, 3, 1); // Spanning multiple columns for the header
grid.add(secretCodeLabel, 0, 1);
grid.add(secretCodeField, 1, 1);
grid.add(filePathLabel, 0, 2);
grid.add(filePathField, 1, 2);
grid.add(chooseFileButton, 2, 2);
grid.add(keySizeLabel, 0, 3);
grid.add(keySizeComboBox, 1, 3);
grid.add(encryptButton, 0, 4, 3, 1); // Spanning multiple columns for the button

// Set scene
Scene scene = new Scene(grid, 500, 250);
scene.getStylesheets().add(getClass().getResource("style.css").toExternalForm());
primaryStage.setScene(scene);
primaryStage.show();
}

private void chooseFile(Stage primaryStage) {
    FileChooser fileChooser = new FileChooser();
    fileChooser.setTitle("Select File");
    File selectedFile = fileChooser.showOpenDialog(primaryStage);
    if (selectedFile != null) {
        filePathField.setText(selectedFile.getAbsolutePath());
    }
}

static String generateKey(String str, String key) {
    int x = str.length();

    for (int i = 0; i < x; i++) {
        if (x == i)

```

```

        i = 0; //reaching the end of key, so start again
        if (key.length() == str.length())
            break; //break as soon as key length equals string length
        key += (key.charAt(i)); //else keep adding characters of the given keyword
    }
    return key.toUpperCase();
}
static String cipherText(String str, String key) {
    String cipher_text = "";

    for (int i = 0; i < str.length(); i++) {
        char c = str.charAt(i);
        char k = key.charAt(i); // Use key characters cyclically

        if (Character.isLetter(c)) {
            if (Character.isUpperCase(c)) {
                int x = (c - 'A' + k - 'A') % 26;
                cipher_text += (char) (x + 'A');
            } else {
                int x = (c - 'a' + k - 'A') % 26; // Always use uppercase key
                cipher_text += (char) (x + 'a');
            }
        }

        } else if (Character.isDigit(c)) {
            int shiftedDigit = ((c - '0') + 3) % 10;
            cipher_text += (char) (shiftedDigit + '0');
        } else {
            cipher_text += c;
        }
    }
    return cipher_text;
}

static String originalText(String cipher_text, String key) {
    String orig_text = "";

    for (int i = 0; i < cipher_text.length(); i++) {
        char c = cipher_text.charAt(i);
        char k = key.charAt(i); // Use key characters cyclically
        if (Character.isLetter(c)) {
            if (Character.isUpperCase(c)) {
                int x = (c - 'A' - (k - 'A') + 26) % 26;
                orig_text += (char) (x + 'A');
            } else {
                int x = (c - 'a' - (k - 'A') + 26) % 26; // Always use uppercase key
                orig_text += (char) (x + 'a');
            }
        }
        } else if (Character.isDigit(c)) {
            int shiftedDigit = ((c - '0') - 3 + 10) % 10;
            orig_text += (char) (shiftedDigit + '0');
        } else {
            orig_text += c;
        }
    }
}

```

```

    }
    }
    return orig_text;
}

// Choose output file extension based on input file extension
public static String chooseOutputFilePath(String inputFilePath, boolean isEncryption) {
    String outputFilePath = inputFilePath.substring(0,inputFilePath.lastIndexOf('.'));
    int dotIndex = inputFilePath.lastIndexOf('.');
    if (dotIndex != -1) {
        String extension = inputFilePath.substring(dotIndex + 1);
        if (isEncryption) {
            // Append appropriate encryption extension based on input file extension
            switch (extension.toLowerCase()) {
                case "docx":
                    outputFilePath += "_encrypted.docx";
                    break;
                case "xlsx":
                    outputFilePath += "_encrypted.xlsx";
                    break;
                case "doc":
                    outputFilePath += "_encrypted.doc";
                    break;
                case "txt":
                    outputFilePath += "_encrypted.txt";
                    break;
                case "csv":
                    outputFilePath += "_encrypted.csv";
                    break;
                default:
                    outputFilePath += "_encrypted.dat"; // Default extension
            }
        } else {
            // Remove existing extension and append appropriate decryption extension
            //outputFilePath = inputFilePath.substring(0, dotIndex);
            switch (extension.toLowerCase()) {
                case "docx":
                case "xlsx":
                case "doc":
                case "txt":
                case "csv":
                    // Remove "_encrypted" suffix if present
                    outputFilePath = outputFilePath.replace("_encrypted", "_decrypted.");
                    outputFilePath += extension;
                    break;
                default:
                    outputFilePath += "_decrypted.dat"; // Default extension
            }
        }
    } else {
        // No extension found in input file path
        if (isEncryption) {

```

```

        outputFilePath += "_encrypted.dat"; // Default extension
    } else {
        outputFilePath += "_decrypted.dat"; // Default extension
    }
}
return outputFilePath;
}

private void encryptFile() {
    // Get user input
    CharSequence secretCodeChars = secretCodeField.getCharacters();
    String secretCode = secretCodeChars.toString();
    String filePath = filePathField.getText();
    int keySize = keySizeComboBox.getValue();
    if (!secretCode.matches("[a-zA-Z]*")) {
        // Display an alert box to the user
        Alert alert = new Alert(Alert.AlertType.ERROR);
        alert.setTitle("Error");
        alert.setHeaderText("Invalid Input");
        alert.setContentText("Secret code should only contain letters.");
        alert.showAndWait();

        // Reset the form field
        secretCodeField.clear();
        return; // Exit method to prevent further execution
    }

    // Call the existing encryptFile method with user input
    try {
        Class.forName("com.mysql.cj.jdbc.Driver");
        Connection con=DriverManager.getConnection("jdbc:mysql://localhost:3306/mds","root","roy1968");
        System.out.println("Successfully Connected to Database!");

        String key=generateKey("AsMiTa2001",secretCode);
        String cipher_text = cipherText("AsMiTa2001", key);
        encryptFile(cipher_text, filePath, keySize, con);
        System.out.println("Encryption successful.");
        con.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
}

public static void encryptFile(String secretCode, String filePath, int keySize, Connection con) throws Exception {
    // Read file content
    byte[] fileContent = readFileContent(filePath);

    // Generate a secret key with the desired key size
    SecretKey secretKey = generateSecretKey(keySize);

    // Generate a random IV

```

```

byte[] iv = generateIV();

// Perform encryption on the file content with the secret key and IV
byte[] encryptedBytes = performEncryption(fileContent, secretKey, iv);

// Choose output file extension based on input file extension
String outputFilePath = chooseOutputFilePath(filePath, true);

// Write encrypted bytes to a file
writeToFile(outputFilePath, encryptedBytes);

// Convert secret key, IV to Base64 strings
String base64SecretKey = Base64.getEncoder().encodeToString(secretKey.getEncoded());
String base64IV = Base64.getEncoder().encodeToString(iv);

System.out.println("File encrypted successfully. Encrypted file saved as: " + outputFilePath);
System.out.println("Secret Key (Base64): " + Base64.getEncoder().encodeToString(secretKey.getEncoded()));
System.out.println("IV (Base64): " + Base64.getEncoder().encodeToString(iv));

// Prepare SQL INSERT statement
String insertQuery = "INSERT INTO cryptify1 (secret_key, iv, secret_code, encrypted_file) VALUES (?, ?, ?,
?);";
PreparedStatement pstmt = con.prepareStatement(insertQuery);

// Set values in the prepared statement
pstmt.setString(1, base64SecretKey);
pstmt.setString(2, base64IV);
pstmt.setString(3, secretCode);
pstmt.setBytes(4, encryptedBytes);

// Execute INSERT statement to store data in the database
pstmt.executeUpdate();

// Close resources
pstmt.close();
displayEncryptionInfo(outputFilePath, base64SecretKey, base64IV);
}

private static void displayEncryptionInfo(String outputFilePath, String secretKey, String iv) {
    Alert alert = new Alert(Alert.AlertType.INFORMATION);
    alert.setTitle("Encryption Successful");
    alert.setHeaderText("File Encrypted Successfully");
    alert.setContentText("File Name: " + outputFilePath + "\n\n" +
        "Secret Key: " + secretKey + "\n\n" +
        "Initialization Vector (IV): " + iv);
    alert.showAndWait();
}

// Read file content into a byte array
public static byte[] readFileContent(String filePath) throws IOException {

```

```

File file = new File(filePath);
byte[] fileContent = new byte[(int) file.length()];
FileInputStream fis = new FileInputStream(file);
fis.read(fileContent);
fis.close();
return fileContent;
}

// Perform encryption on the input bytes with the secret key and IV
public static byte[] performEncryption(byte[] inputBytes, SecretKey secretKey, byte[] iv) throws Exception {
    // Encryption
    Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
    cipher.init(Cipher.ENCRYPT_MODE, secretKey, new IvParameterSpec(iv));
    return cipher.doFinal(inputBytes);
}

// Generate a secret key with the desired key size
public static SecretKey generateSecretKey(int keySize) throws Exception {
    KeyGenerator keyGenerator = KeyGenerator.getInstance("AES");
    keyGenerator.init(keySize);
    return keyGenerator.generateKey();
}

// Generate a random IV (Initialization Vector)
public static byte[] generateIV() {
    byte[] iv = new byte[16]; // IV size is typically 16 bytes for AES
    new java.security.SecureRandom().nextBytes(iv);
    return iv;
}

// Write byte array to a file
public static void writeToFile(String filePath, byte[] data) throws IOException {
    FileOutputStream fos = new FileOutputStream(filePath);
    fos.write(data);
    fos.close();
}

public static void main(String[] args) throws Exception {
    launch(args);
}
}

```

Decryption Code

```
package application;
import javafx.application.Application;
import javafx.geometry.Insets;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.layout.GridPane;
import javafx.stage.FileChooser;
import javafx.stage.Stage;
import javax.crypto.*;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.SecretKeySpec;
import java.io.*;
import java.sql.*;
import java.util.*;

public class Main1 extends Application {
    private TextField secretCodeField;
    private TextField encryptedFilePathField;
    private TextField secretKeyField;
    private TextField ivField;

    @Override
    public void start(Stage primaryStage) throws Exception {
        primaryStage.setTitle("Asmita Roy Mondal");
        // Create header label
        Label headerLabel = new Label("CRYPTIFY DECRYPTION");
        headerLabel.setStyle("-fx-font-size: 24px; -fx-font-weight: bold; -fx-background-color: #7B68EE; -fx-padding: 10px; -fx-text-fill: white; -fx-background-radius: 5px;");
        Label secretCodeLabel = new Label("Secret Code:");
        // Replace the existing TextField with PasswordField
        PasswordField secretCodeField = new PasswordField();
        secretCodeField.setPromptText("Enter Secret Code");
        this.secretCodeField = secretCodeField;
        Label encryptedFilePathLabel = new Label("Encrypted File Path:");
        encryptedFilePathField = new TextField();
        encryptedFilePathField.setPromptText("Select Encrypted File");
        Label secretKeyLabel = new Label("Secret Key (Base64):");
        secretKeyField = new TextField();
        secretKeyField.setPromptText("Enter Secret Key");
        Label ivLabel = new Label("Initialization Vector (Base64):");
        ivField = new TextField();
        ivField.setPromptText("Enter IV");
        Button chooseFileButton = new Button("Choose Encrypted File");
        chooseFileButton.setOnAction(e -> chooseEncryptedFile(primaryStage));
    }
}
```

```

Button decryptButton = new Button("Decrypt");
decryptButton.setOnAction(e -> decryptFile());
// Create layout
GridPane grid = new GridPane();
grid.setPadding(new Insets(20));
grid.setVgap(10);
grid.setHgap(10);
grid.add(headerLabel, 0, 0, 3, 1); // Spanning multiple columns for the header
grid.add(secretCodeLabel, 0, 1);
grid.add(secretCodeField, 1, 1);
grid.add(encryptedFilePathLabel, 0, 2);
grid.add(encryptedFilePathField, 1, 2);
grid.add(chooseFileButton, 2, 2);
grid.add(secretKeyLabel, 0, 3);
grid.add(secretKeyField, 1, 3);
grid.add(ivLabel, 0, 4);
grid.add(ivField, 1, 4);
grid.add(decryptButton, 0, 5, 3, 1); // Spanning multiple columns for the button
// Set scene
Scene scene = new Scene(grid, 650, 300);
scene.getStylesheets().add(getClass().getResource("style.css").toExternalForm());
primaryStage.setScene(scene);
primaryStage.show();
}

private void chooseEncryptedFile(Stage primaryStage) {
    FileChooser fileChooser = new FileChooser();
    fileChooser.setTitle("Select Encrypted File");
    File selectedFile = fileChooser.showOpenDialog(primaryStage);
    if (selectedFile != null) {
        encryptedFilePathField.setText(selectedFile.getAbsolutePath());
    }
}

private void decryptFile() {
    // Get user input
    CharSequence secretCodeChars = secretCodeField.getCharacters();
    String secretCode = secretCodeChars.toString();
    String encryptedFilePath = encryptedFilePathField.getText();
    String base64Key = secretKeyField.getText();
    String base64IV = ivField.getText();
    // Call the decryptFile method with user input
    try {
        Scanner scanner = new Scanner(System.in);
        Class.forName("com.mysql.cj.jdbc.Driver");
        Connection con = DriverManager.getConnection("jdbc:mysql://localhost:3306/mds", "root", "roy1968");
        System.out.println("Successfully Connected to Database!");
        decryptFile(scanner, con, secretCode, encryptedFilePath, base64Key, base64IV);
        con.close();
    } catch (Exception e) {
        e.printStackTrace();
    }
}
}

```



```

    public static void decryptFile(Scanner scanner, Connection con, String secretCode, String encryptedFilePath,
String base64Key, String base64IV) throws Exception {
    // Input secret key
    byte[] keyBytes = Base64.getDecoder().decode(base64Key);
    SecretKey secretKey = new SecretKeySpec(keyBytes, "AES");
    // Input IV
    byte[] iv = Base64.getDecoder().decode(base64IV);
    String selectQuery = "SELECT secret_code FROM cryptify1 WHERE secret_key = ? AND iv = ?";
    PreparedStatement pstmt = con.prepareStatement(selectQuery);
    pstmt.setString(1, base64Key);
    pstmt.setString(2, base64IV);
    ResultSet rs = pstmt.executeQuery();

    if(rs.next()) {
        String text = rs.getString("secret_code");
        String key = generateKey(text, secretCode);
        String original_text = originalText(text, key);
        if (original_text.equals("AsMiTa2001")) {
            // Read encrypted bytes from file
            byte[] encryptedBytes = readFileContent(encryptedFilePath);
            // Perform decryption on the encrypted bytes
            byte[] decryptedBytes = performDecryption(encryptedBytes, secretKey, iv);
            // Choose output file extension based on input file extension
            String outputFilePath = chooseOutputFilePath(encryptedFilePath, false);
            // Write decrypted bytes to a file
            writeToFile(outputFilePath, decryptedBytes);
            System.out.println("File decrypted successfully. Decrypted file saved as: " + outputFilePath);
            displayDecryptionSuccess(outputFilePath);
        } else {
            System.out.println("OOPS! Wrong Code");
            displayWrongCodeError1();
        }
    }
    else
    {
        displayWrongCodeError();
    }
}

private static void displayDecryptionSuccess(String outputFilePath) {
    Alert alert = new Alert(Alert.AlertType.INFORMATION);
    alert.setTitle("Decryption Successful");
    alert.setHeaderText("File Decrypted Successfully");
    // int dotIndex = outputFilePath.lastIndexOf("\\");
    // String path=outputFilePath.substring(dotIndex+1);
    alert.setContentText("Decrypted file saved as:\n" + outputFilePath+ "\n\n"+ "\r\n"); // Set content text with the
output file path
    alert.showAndWait();
}

private static void displayWrongCodeError() {
    Alert alert = new Alert(Alert.AlertType.ERROR);
    alert.setTitle("Decryption Error");
}

```

```

        alert.setHeaderText("Wrong Code");
        alert.setContentText("OOPS! Wrong Code");
        alert.showAndWait();
    }
    private static void displayWrongCodeError1() {
        Alert alert = new Alert(Alert.AlertType.ERROR);
        alert.setTitle("Decryption Error");
        alert.setHeaderText("Wrong Key or IV");
        alert.setContentText("OOPS! Please Check Your Codes");
        alert.showAndWait();
    }

    // Read file content into a byte array
    public static byte[] readFileContent(String filePath) throws IOException {
        File file = new File(filePath);
        byte[] fileContent = new byte[(int) file.length()];
        FileInputStream fis = new FileInputStream(file);
        fis.read(fileContent);
        fis.close();
        return fileContent;
    }

    // Perform decryption on the input bytes with the secret key and IV
    public static byte[] performDecryption(byte[] encryptedBytes, SecretKey secretKey, byte[] iv) throws Exception {
        // Decryption
        Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
        cipher.init(Cipher.DECRYPT_MODE, secretKey, new IvParameterSpec(iv));
        return cipher.doFinal(encryptedBytes);
    }

    // Generate key based on secret code and input key
    public static String generateKey(String str, String key) {
        int x = str.length();
        for (int i = 0; i < key.length(); i++) {
            if (x == i)
                i = 0; // reaching the end of key, so start again
            if (key.length() == str.length())
                break; // break as soon as key length equals string length
            key += (key.charAt(i)); // else keep adding characters of the given keyword
        }
        return key.toUpperCase();
    }

    static String originalText(String cipher_text, String key) {
        String orig_text = "";
        for (int i = 0; i < cipher_text.length(); i++) {
            char c = cipher_text.charAt(i);
            char k = key.charAt(i); // Use key characters cyclically
            if (Character.isLetter(c)) {
                if (Character.isUpperCase(c)) {
                    int x = (c - 'A' - (k - 'A') + 26) % 26;
                    orig_text += (char) (x + 'A');
                } else {
                    int x = (c - 'a' - (k - 'A') + 26) % 26; // Always use uppercase key
                    orig_text += (char) (x + 'a');
                }
            }
        }
    }

```

```

    }
    } else if (Character.isDigit(c)) {
        int shiftedDigit = ((c - '0') - 3 + 10) % 10;
        orig_text += (char) (shiftedDigit + '0');
    } else {
        orig_text += c;
    }
}
return orig_text;
}
// Choose output file extension based on input file extension
public static String chooseOutputFilePath(String inputFilePath, boolean isEncryption) {
    String outputFilePath = inputFilePath.substring(0, inputFilePath.lastIndexOf('.'));
    int dotIndex = inputFilePath.lastIndexOf('.');
    if (dotIndex != -1) {
        String extension = inputFilePath.substring(dotIndex + 1);
        if (isEncryption) {
            // Append appropriate encryption extension based on input file extension
            switch (extension.toLowerCase()) {
                case "docx":
                    outputFilePath += "_encrypted.docx";
                    break;
                case "xlsx":
                    outputFilePath += "_encrypted.xlsx";
                    break;
                case "doc":
                    outputFilePath += "_encrypted.doc";
                    break;
                case "txt":
                    outputFilePath += "_encrypted.txt";
                    break;
                case "csv":
                    outputFilePath += "_encrypted.csv";
                    break;
                default:
                    outputFilePath += "_encrypted.dat"; // Default extension
            }
        } else {
            // Remove existing extension and append appropriate decryption extension
            // outputFilePath = inputFilePath.substring(0, dotIndex);
            switch (extension.toLowerCase()) {
                case "docx":
                case "xlsx":
                case "doc":
                case "txt":
                case "csv":
                    // Remove "_encrypted" suffix if present
                    outputFilePath = outputFilePath.replace("_encrypted", "_decrypted.");
                    outputFilePath += extension;
                    break;
                default:
                    outputFilePath += "_decrypted.dat"; // Default extension
            }
        }
    }
}

```

```

    }
}
} else {
    // No extension found in input file path
    if (isEncryption) {
        outputFilePath += "_encrypted.dat"; // Default extension
    } else {
        outputFilePath += "_decrypted.dat"; // Default extension
    }
}
}
return outputFilePath;
}
// Write byte array to a file
public static void writeToFile(String filePath, byte[] data) throws IOException {
    FileOutputStream fos = new FileOutputStream(filePath);
    fos.write(data);
    fos.close();
}
public static void main(String[] args) throws Exception {
    launch(args);
}
}
}

```

CSS Styling

```

.root {
    -fx-background-color: lavender;
}
.text-field {
    -fx-background-color: #FFFFFF; /* White background */
    -fx-text-inner-color: black; /* Black text */
    -fx-border-color: purple;
    -fx-border-width: 1px;
    -fx-border-radius: 5px;
}
.button {
    -fx-background-color: #7B68EE; /* Lavender background */
    -fx-text-fill: white; /* White text */
    -fx-font-weight: bold; /* Bold font */
    -fx-font-size: 15px;
}
.button:hover {
    -fx-background-color: #6A5ACD; /* Darker lavender background on hover */
}

```

```

.label {
    -fx-text-fill: purple;
    -fx-font-size: 16px;
    -fx-font-family: "Helvetica"; /* Change the font family */
}

.combo-box {
    -fx-background-color: white;
    -fx-text-fill: black;
    -fx-font-size: 15px;
}

```

Database and Table Creation

```

CREATE DATABASE mds;
USE mds;

CREATE TABLE cryptify1 (
    id INT AUTO_INCREMENT PRIMARY KEY,
    secret_key VARCHAR(200),
    iv VARCHAR(200),
    secret_code VARCHAR(200),
    encrypted_file LONGBLOB
);
select * from cryptify1;

```

module-info.java

```

module Try {
    requires javafx.controls;
    requires java.sql;
    requires javafx.graphics;

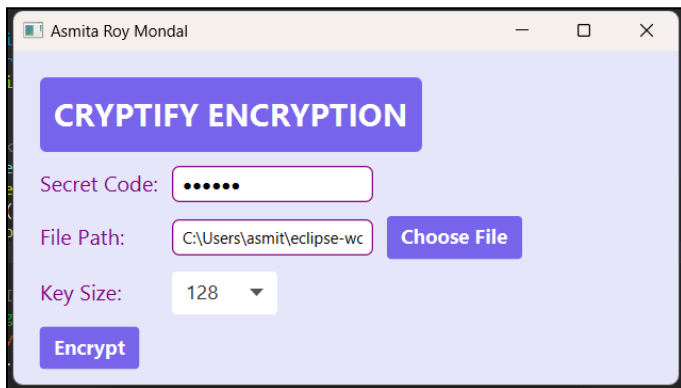
    opens application to javafx.graphics, javafx.fxml;
}

```

CHAPTER 8: RESULTS AND DISCUSSIONS

Running the Project

1. Encryption Interface:



Asmita Roy Mondal

CRYPTIFY ENCRYPTION

Secret Code:

File Path: C:\Users\asmit\workspace\ChristJava\test_encrypted.csv **Choose File**

Key Size: 128

Encrypt

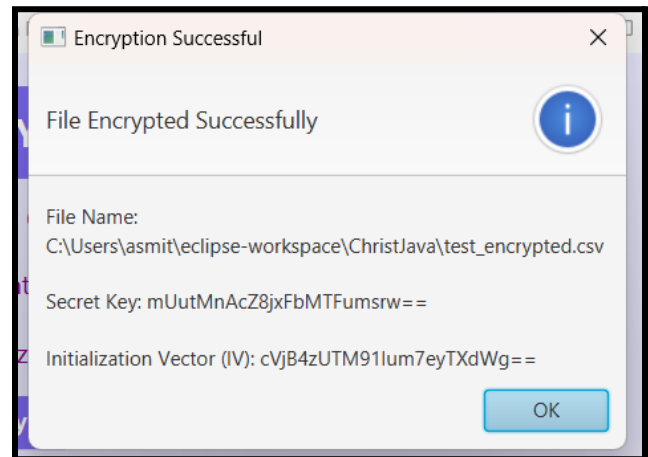
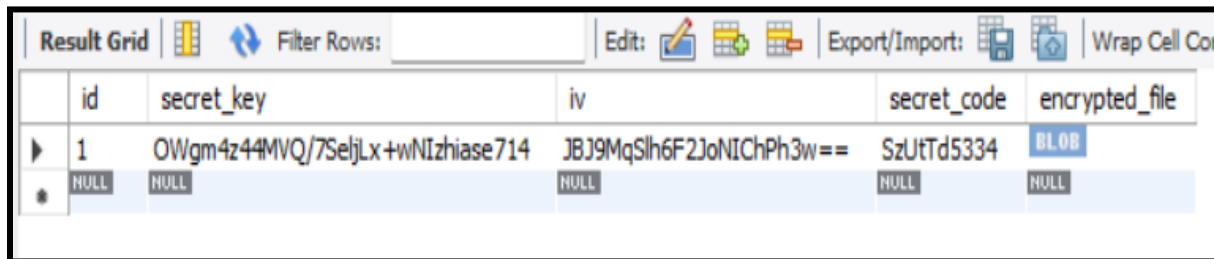


Figure 4: Interface and Success Alert Box

2. Database Storage:



	id	secret_key	iv	secret_code	encrypted_file
▶	1	OWgm4z44MVQ/7SeljLx+wNIzhiase714	JBj9MqSlh6F2JoNIChPh3w==	SzUtTd5334	BLOB
*	NULL	NULL	NULL	NULL	NULL

Figure 5: Table in Database

3. Decryption Interface:

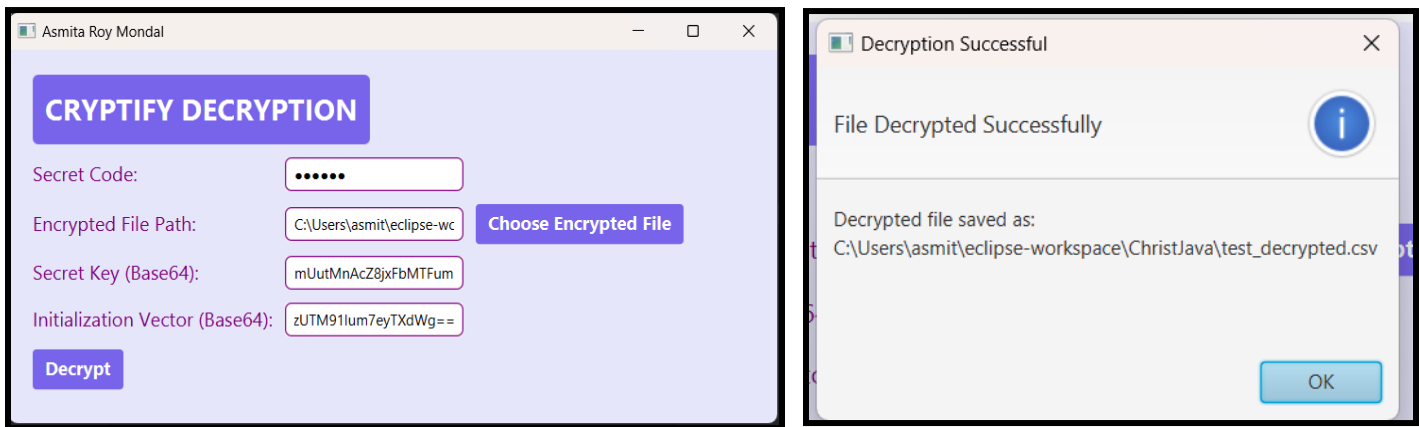


Figure 6: Decryption Interface and Success Message

4. Error if Input is Wrong:

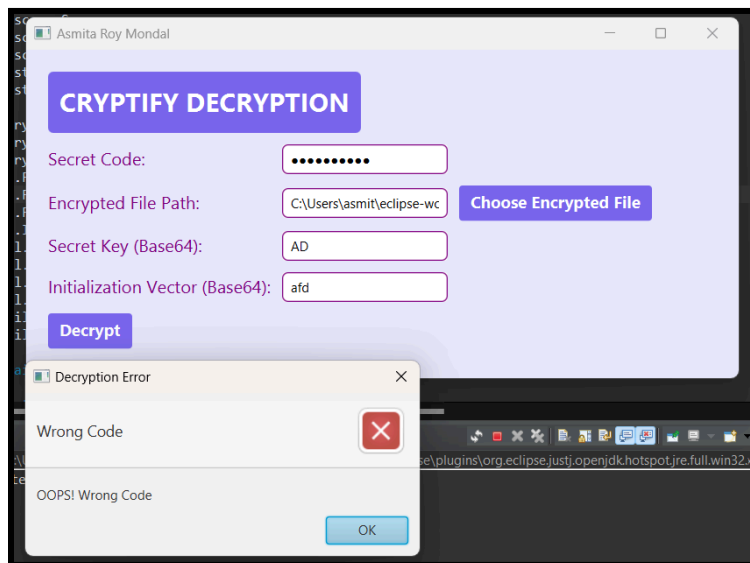


Figure 7: Error Display with wrong Key or IV Input

5. Output of Two Algorithms:

[illegible]

1	ObjectID2	Country	ISO2	ISO3	Indicator	Unit	Source	CTS_Code	CTS_Name	CTS_Full_I	Industry	Gas_Type	Scale	F2010
2	1	Advanced Economies	AETMP		Annual gre	Million me	Organisati	ECNGA	Greenhou	Environme	Agriculture	Carbon dic	Units	194.916
3	2	Advanced Economies	AETMP		Annual gre	Million me	Organisati	ECNGA	Greenhou	Environme	Agriculture	Fluorinate	Units	0.92304
4	3	Advanced Economies	AETMP		Annual gre	Million me	Organisati	ECNGA	Greenhou	Environme	Agriculture	Greenhou	Units	1356.499
5	4	Advanced Economies	AETMP		Annual gre	Million me	Organisati	ECNGA	Greenhou	Environme	Agriculture	Methane	Units	609.180
6	5	Advanced Economies	AETMP		Annual gre	Million me	Organisati	ECNGA	Greenhou	Environme	Agriculture	Nitrous ox	Units	551.476
7	6	Advanced Economies	AETMP		Annual gre	Million me	Organisati	ECNGA	Greenhou	Environme	Constructi	Carbon dic	Units	328.299
8	7	Advanced Economies	AETMP		Annual gre	Million me	Organisati	ECNGA	Greenhou	Environme	Constructi	Fluorinate	Units	2.0635
9	8	Advanced Economies	AETMP		Annual gre	Million me	Organisati	ECNGA	Greenhou	Environme	Constructi	Greenhou	Units	335.481
10	9	Advanced Economies	AETMP		Annual gre	Million me	Organisati	ECNGA	Greenhou	Environme	Constructi	Methane	Units	1.24984
11	10	Advanced Economies	AETMP		Annual gre	Million me	Organisati	ECNGA	Greenhou	Environme	Constructi	Nitrous ox	Units	3.3853
12	11	Advanced Economies	AETMP		Annual gre	Million me	Organisati	ECNGA	Greenhou	Environme	Electricity	Carbon dic	Units	4527.64
13	12	Advanced Economies	AETMP		Annual gre	Million me	Organisati	ECNGA	Greenhou	Environme	Electricity	Fluorinate	Units	2.53819
14	13	Advanced Economies	AETMP		Annual gre	Million me	Organisati	ECNGA	Greenhou	Environme	Electricity	Greenhou	Units	4684.9
15	14	Advanced Economies	AETMP		Annual gre	Million me	Organisati	ECNGA	Greenhou	Environme	Electricity	Methane	Units	109.292
16	15	Advanced Economies	AETMP		Annual gre	Million me	Organisati	ECNGA	Greenhou	Environme	Electricity	Nitrous ox	Units	45.4945
17	16	Advanced Economies	AETMP		Annual gre	Million me	Organisati	ECNGA	Greenhou	Environme	Manufact	Carbon dic	Units	2562.46
18	17	Advanced Economies	AETMP		Annual gre	Million me	Organisati	ECNGA	Greenhou	Environme	Manufact	Fluorinate	Units	72.5247
19	18	Advanced Economies	AETMP		Annual gre	Million me	Organisati	ECNGA	Greenhou	Environme	Manufact	Greenhou	Units	2717.7
20	19	Advanced Economies	AETMP		Annual gre	Million me	Organisati	ECNGA	Greenhou	Environme	Manufact	Methane	Units	28.1358
21	20	Advanced Economies	AETMP		Annual gre	Million me	Organisati	ECNGA	Greenhou	Environme	Manufact	Nitrous ox	Units	54.5877
22	21	Advanced Economies	AETMP		Annual gre	Million me	Organisati	ECNGA	Greenhou	Environme	Mining	Carbon dic	Units	310.602
23	22	Advanced Economies	AETMP		Annual gre	Million me	Organisati	ECNGA	Greenhou	Environme	Mining	Fluorinate	Units	0.05146
24	23	Advanced Economies	AETMP		Annual gre	Million me	Organisati	ECNGA	Greenhou	Environme	Mining	Greenhou	Units	617.572
25	24	Advanced Economies	AETMP		Annual gre	Million me	Organisati	ECNGA	Greenhou	Environme	Mining	Methane	Units	304.284

Figure 8: Encrypted vs Decrypted CSV Text Example

Observations

A comparison was performed on the files that could be used for the project and the following was observed:

- Key Size: 128 bits
- Secret Code: SHILADITYA

TYPE OF FILE	ENCRYPTION TIME	DECRYPTION TIME
CSV	2195297200 nanoseconds	2466278400 nanoseconds
TXT	2695256400 nanoseconds	2920334700 nanoseconds
DOCX	2350019500 nanoseconds	2827815200 nanoseconds
XLSX	2162265900 nanoseconds	2522509500 nanoseconds

Table 1: Comparison of Time for Different File Types

1. CSV files took least time to perform all processes while TXT files took the most. Even though the difference in time is less, it still has a significant impact when the size of data is even bigger.
2. This difference can be accounted for by the presence of free-form text in TXT files and structured, well defined data in CSV or XLSX files which makes it easier to find patterns and make the process faster. TXT files require more time for parsing and processing as compared to the compressed, patterned CSV files.

CHAPTER 9: CONCLUSION

In conclusion, the study has explored various encryption techniques, including classical methods like Caesar and Vigenère ciphers, as well as modern symmetric encryption algorithms such as AES. Symmetric encryption algorithms like AES provide robust security through complex mathematical operations and large key spaces, making them suitable for securing sensitive and large data in today's digital age. Caesar and Vigenère ciphers have a simple implementation to secure the personalized Secret Codes that the user gives.

Only when the person at the decryption ends knows the Secret Key, Secret Code and Iv and has the correct encrypted file will he/she be able to get back the original file. If any one of them goes wrong, there are no chances of decryption. It was also found that AES encryption worked the best for larger datasets. RSA is not suitable for the same as the time complexity increases.

Further, CSV files got encrypted and decrypted the fastest while TXT files took more time. This may be because TXT files contain more free-flowing text while CSV has structured patterns that can lead to a faster encryption process.

Limitations

1. Caesar and Vigenère ciphers used for Secret Code encryption are vulnerable to frequency analysis attacks, where an attacker can analyze the frequency distribution of characters in the ciphertext to deduce the encryption key or plaintext.
2. The project is not scalable yet. It has handled large data but due to the unavailability of larger resources, it has not been tested on larger datasets.
3. It works on a set list of file types. Types like Images, or PDF are not supported.

Future Scope

1. To incorporate more file types such as audio, images and pdf thus streamlining the process for better usability and scalability.
2. To create a deployable web application that can be accessed by users globally.
3. To use more encryption layers, algorithms and techniques so as to make the process more robust and defiant to attacks.

REFERENCES

- [1] Lu, Z., & Mohamed, H. (2021). A complex encryption system design implemented by AES. *Journal of Information Security*, 12(2), 177-187. DOI: 10.4236/jis.2021.122009
- [2] Tayde, S., & Siledar, S. (2015). File encryption, decryption using AES algorithm in android phone. *International Journal of Advanced Research in computer science and software engineering*, 5(5).
- [3] Amola, A., & Saha, S. (2016). Android Security: Security using Encryption (AES).
- [4] S. Saudagar, N. Kamtalwar, H. Karadbhajne, M. Karmarkar, H. Kendre and O. Ketkar, "File Encryption-Decryption using Java," *2023 International Conference on Intelligent Data Communication Technologies and Internet of Things (IDCIoT)*, Bengaluru, India, 2023, pp. 855-859, doi: 10.1109/IDCIoT56793.2023.10053514. keywords: {Java;Protocols;XML;Public key;Media;Encryption;Virtual private networks;Encryption;Decryption;Java;File;Blowfish;Caesar Cipher;Data Encryption Standard},
- [5] Lemma, Abraham & Bravo, Maribel & Mehari, Gebremedhn. (2015). Performance Analysis on the Implementation of Data Encryption Algorithms Used in Network Security. 2279-0764.
- [6] Salama, Dr-Diaa & Abd elkader, Hatem & Hadhoud, Mohiy M.. (2009). Performance Evaluation of Symmetric Encryption Algorithms. Communications of the IBIMA. 10.
- [7] Saraf, K. R., Jagtap, V. P., & Mishra, A. K. (2014). Text and image encryption decryption using advanced encryption standard. *International Journal of Emerging Trends & Technology in Computer Science (IJETTCS)*, 3(3), 118-126.