“File Encryption and Decryption”

A Project Report Submitted in the partial fulfillment of the requirements of the course titled

“Problem Solving Through Programming (JAVA)”

BACHELOR OF TECHNOLOGY

In

DEPARTMENT OF FRESHMAN ENGINEERING

By

K. Rishitha Chowdary	**2520030202**
Ch. Shravya	**2520030322**
C. Thanvi	**2520030214**
Y. Bhaarathi	**2520030211**
R. Pravalika	**2520030402**
Sri Bhargavi	**2520040048**

Under the Esteemed Guidance of

K. Ramakanth

Assisant.Prof

Department of Freshman Engineering

Koneru Lakshmaiah Education Foundation

(Deemed to be University estd. u/s. 3 of the UGC Act, 1956)
Off-Campus: Bachupally-Gandimaisamma Road, Bowrampet, Hyderabad, Telangana - 500 043.
Phone No: 7815926816, www.klh.edu.in

K L (Deemed to be) University

DEPARTMENT OF FRESHMAN ENGINEERING

Declaration

The Project Report entitled **"File Encryption and Decryption"** is a record of Bonafide work of K. Rishitha Chowdary (2520030202), Ch. Shravya (2520030322), C. Thanvi (2520030214),Y. Bhaarathi (2520030211) ,R. Pravalika (2520030402), Sri Bhargvi(2520040048), submitted in partial fulfillment of the requirements of the course titled "Problem Solving Through Programming (JAVA)" under the B.Tech Ist Year Trimester - I program in Department of Freshman Engineering at K L University. The results presented in this report have not been copied from any other department, university, or institute.

K. Rishitha Chowdary 2520030202
Ch. Shravya 2520030322
C. Thanvi 2520030214
Y. Bhaarathi 2520030211
R. Pravalika 2520030402
Sri Bhargavi 2520040048

KL (Deemed to be) University

DEPARTMENT OF FRESHMAN ENGINEERING

CERTIFICATE

This is certify that the java project based report entitled **"File Encryption and Decryption"** is a Bonafide work done and submitted by K. Rishitha Chowdary (2520030202), Ch. Shravya (2520030322), C. Thanvi (2520030214),Y. Bhaarathi (2520030211) ,R. Pravalika (2520030402), Sri Bhargvi(2520040048), in partial fulfillment of the requirements of the course titled "Problem Solving Through Programming (JAVA)" under the B.Tech Ist Year Trimester - I program in Department of Freshman Engineering, K L (Deemed to be University), during the academic year **2025-2026.**

Signature of the Guide

Signature of the Course Coordinator **Signature of the HOD**

ACKNOWLEDGEMENT

The success in this project would not have been possible but for the timely help and guidance rendered by many people. Our wish to express my sincere thanks to all those who has assisted us in one way or the other for the completion of my project.

Our greatest appreciation to my Course Coordinator **Dr Y Ashok**, and my guide K. Ramakanth**,** Department of Freshman Engineering which cannot be expressed in words for his/her tremendous support, encouragement and guidance for this project.

We express our gratitude to **Dr. N. Chaitanya Kumar**, Head of the ***Department for Freshman Engineering*** for providing us with adequate facilities, ways and means by which we are able to complete this project-based Lab.

We thank all the members of teaching and non-teaching staff members, and also who have assisted me directly or indirectly for successful completion of this project. Finally, I sincerely thank my parents, friends and classmates for their kind help and cooperation during my work.

K. Rishitha Chowdary 2520030202
Ch. Shravya 2520030322
C. Thanvi 2520030214
Y. Bhaarathi 2520030211
R. Pravalika 2520030402
Sri Bhargavi 2520040048

Abstract:

In an increasingly digital world, the protection of sensitive information has become a crucial requirement. Every day, individuals and organizations share and store personal files, academic documents, financial records, and confidential data across various devices and online platforms. Without proper security, such data becomes vulnerable to unauthorized access, cyberattacks, malware, and data theft. To address these challenges, this project presents a File Encryption and Decryption System developed using Java, designed to provide a simple, efficient, and secure method for protecting files.

The proposed system uses the Advanced Encryption Standard (AES) in Galois/Counter Mode (AES-GCM), a modern and highly secure encryption method widely used in government, banking, and enterprise-level applications. AES-GCM ensures not only confidentiality but also data integrity and authentication, meaning even the slightest modification of encrypted files can be detected. A secure password-based key derivation mechanism (PBKDF2) is used to generate strong cryptographic keys from user passwords, offering better protection against brute-force and dictionary attacks. The application provides a graphical user interface (GUI) that allows users to select files, enter a password, and perform encryption or decryption with ease.

Overall, this project demonstrates the real-world importance of cryptography and secure software development. It highlights how modern algorithms can be implemented effectively in everyday applications to protect user data. Additionally, it strengthens the student's understanding of Java programming, GUI design (Swing), file handling, exception management, and applied cybersecurity concepts. This work serves as a practical and educational solution for secure file management and showcases the effectiveness of Java in building security-focused applications.

Index

CHAPTER -1 INTRODUCTION

1.1 Background of the Project

In today's world, almost everything we do involves digital data. We store personal photos, academic files, ID proofs, bank documents, medical records, and many other important files on computers, mobile devices, and cloud platforms. As our dependency on digital storage increases, the risk of our information being accessed by the wrong people also increases. Cybercrimes such as hacking, data leakage, identity theft, and ransomware attacks are becoming more common. Because of this, protecting our files has become extremely important.

One of the most effective ways to protect data is encryption. Encryption is a process that converts readable data (plaintext) into an unreadable format (ciphertext). Only someone with the correct password or key can convert it back to the original form (decryption). Even if the encrypted file is stolen or accessed without permission, it will still be useless to the attacker because they cannot read it.

To address this need, we developed the File Encryption and Decryption System using Java. This project uses the AES (Advanced Encryption Standard) algorithm, which is one of the strongest and most trusted encryption methods used worldwide. The AES-GCM mode used in this project not only encrypts the data but also checks whether the file has been modified, providing both security and integrity.

The system allows the user to:

- Choose any file
- Enter a password
- Encrypt the file securely
- Decrypt the file when needed

The GUI (Graphical User Interface) makes the tool easy to use, even for beginners. This project also helps students understand important concepts such as cryptography, Java programming, file handling, and secure software development practices.

Overall, the background of this project highlights the real need for secure file protection in the

modern digital world and explains why encryption is essential. It shows how this project provides a simple yet powerful solution for keeping personal and confidential data safe.

1.2 Problem Statement

In today's digital environment, a significant amount of sensitive information—such as personal documents, academic files, financial records, and confidential organizational data—is stored on computers and portable storage devices. Without adequate protection, these files are highly vulnerable to unauthorized access, data breaches, malware attacks, and physical theft of devices. Many users lack technical knowledge about cybersecurity, and existing encryption tools are either too complex, expensive, or not user-friendly, making secure data handling difficult for everyday users.

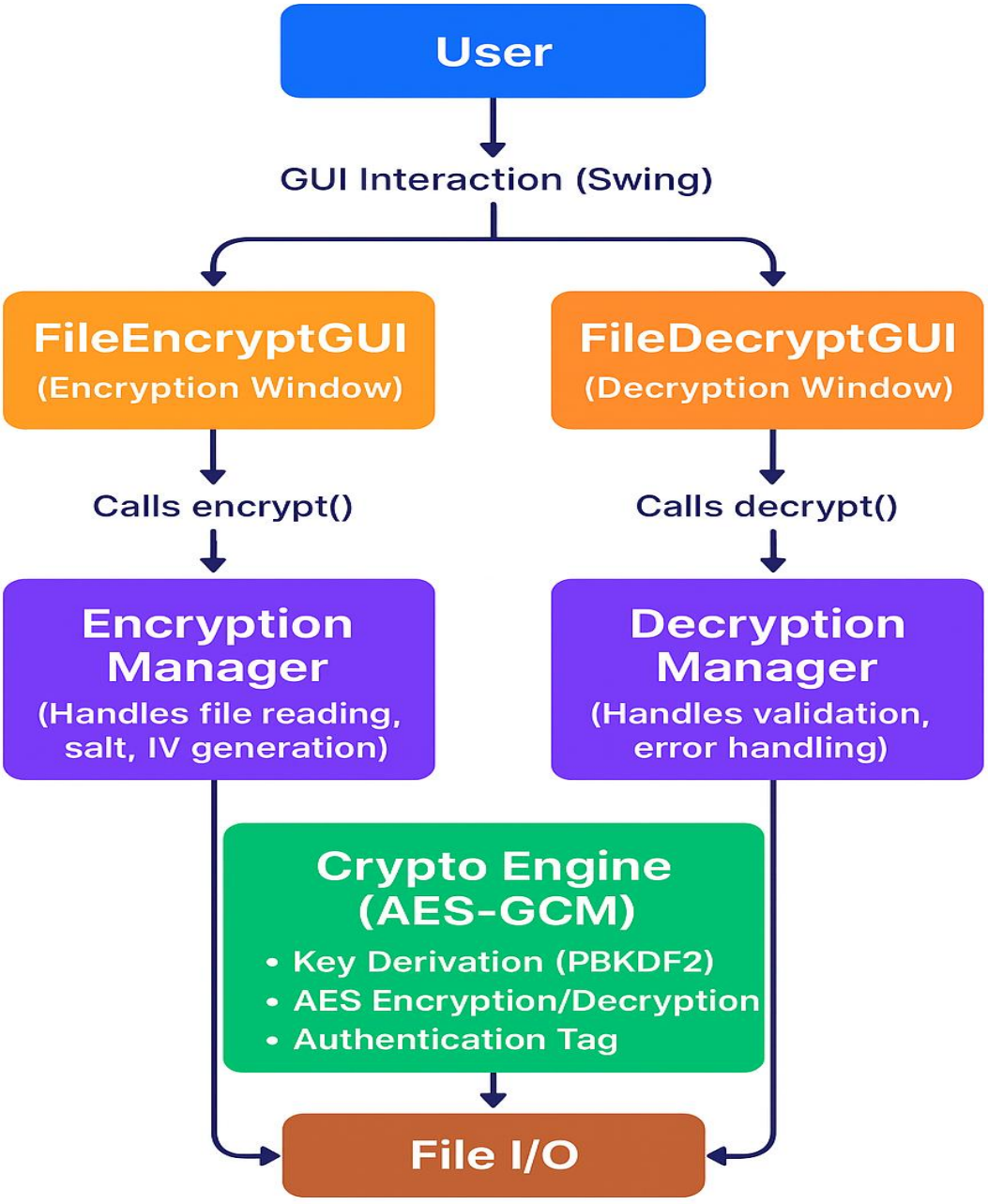
Therefore, there is a strong need for a simple, reliable, and efficient system that ensures the confidentiality and security of personal and sensitive data. This project aims to develop a Java-based file encryption and decryption application using the AES-GCM cryptographic algorithm, which provides both strong security and data integrity. The system enables users to encrypt any type of file using a password and decrypt it only when valid credentials are provided.

The solution provides a graphical user interface (GUI) built with Java Swing, allowing users to securely browse files, enter passwords, encrypt the selected file, and decrypt it whenever required. The system also addresses common security challenges such as key derivation, authentication tag validation, and protection against brute-force or tampering attempts.

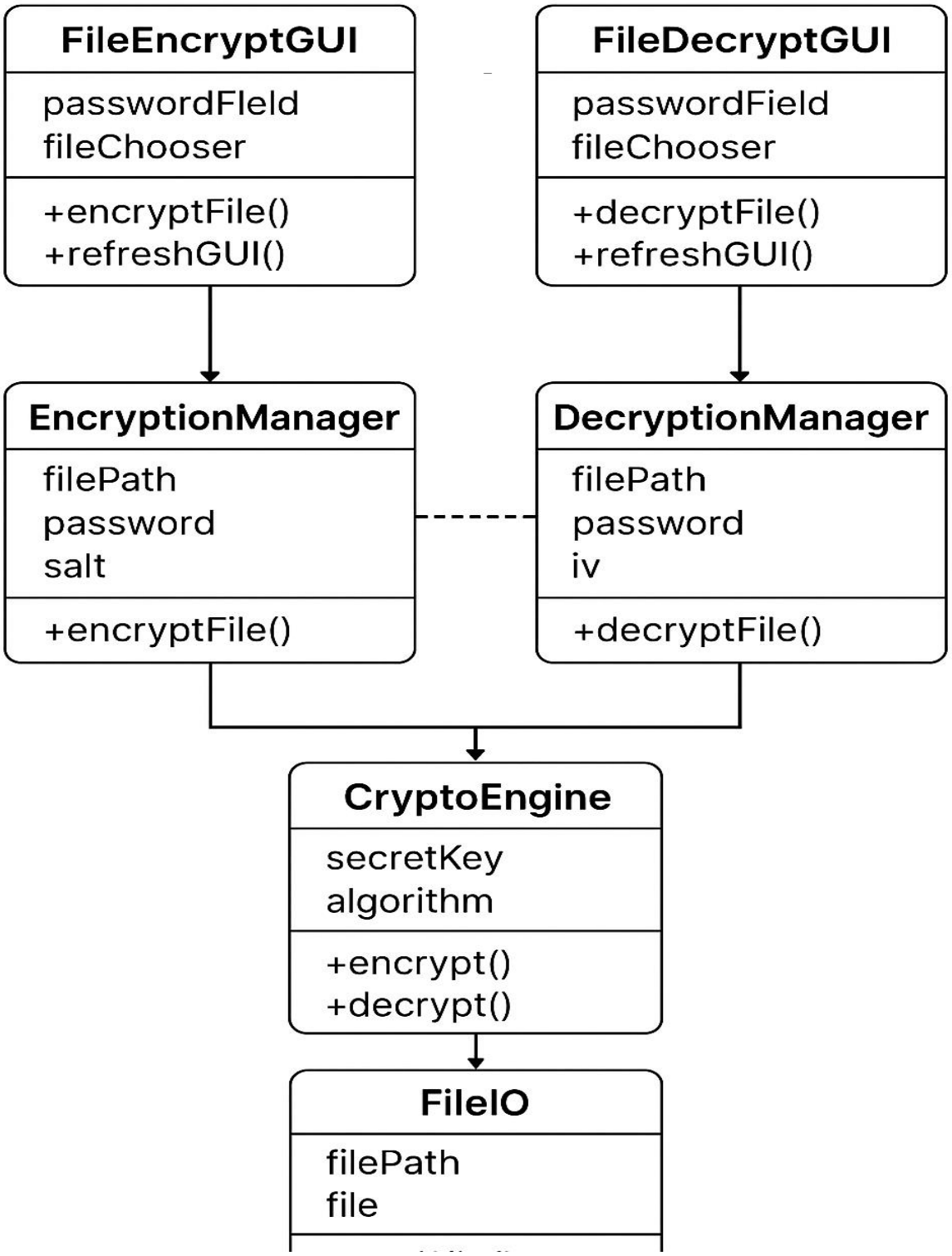
By combining strong cryptographic practices with an easy-to-use interface, the proposed system ensures secure storage and transfers of files for students, professionals, and organizations, thereby reducing risks of data leakage and enhancing overall information security.

CHAPTER -2 SYSTEM ARCHITECTURE

2.1High-level architecture diagram



2.2 Class Diagram



CHAPTER -3 CO's ATTAINMENT

3.1 CO1 Attainment

CO1 Syllabus	CO1 Concepts Included in Project
Apply fundamental programming constructs such as data types, operators, conditional and iterative statements in Java to develop logic-based solutions.	1. Data types: byte[], char[], String, int, long 2. Conditional statements: if, try-catch 3. Loops: Used while reading/writing files 4. Operators: Assignment, comparison, concatenation 5. Exception handling to validate password and file selection 6. GUI input handling (basic event-based logic)

3.1.1 Scenario's for CO1 implementation.

- When the user clicks Encrypt, the program checks:
- Whether the file path is empty
- Whether the password is valid
- If conditions fail, appropriate messages are shown using basic condition checks.
- Reading file data uses loops and basic I/O operations.
- Validating IV length and metadata uses conditional statements.

3.1.2 CO1 code screen shot.

```
// CO1: Basic constants
private static final int SALT_LEN = 16;
private static final int IV_LEN = 12;
private static final int KEY_BITS = 128;
private static final int PBKDF2_ITER = 100_000;
private static final int GCM_TAG_BITS = 128;

// CO1: Input validation
if (id.isEmpty()) { log("Error: Record ID cannot be empty."); return; }
if (pwd.length == 0) { log("Error: Password cannot be empty."); return; }
if (!Files.exists(in)) { log("Error: File not found."); return; }

// CO1: Layout logic
g.insets = new Insets(6, 6, 6, 6);
g.fill = GridBagConstraints.HORIZONTAL;
```

3.2 CO2 Attainment

CO2 Syllabus	CO2 Concepts Included in Project
Design, trace, and optimize algorithms using arrays to solve real-world problems.	1. byte[] array for file content 2. byte[] IV for AES initialization vector 3. byte[] encryptedData for ciphertext 4. byte[] metadataBytes for map storage 5. Password taken as char[]

3.2.1 Scenario's for CO2 implementation.

- Encryption process uses multiple byte arrays to efficiently store and process file data.
- Decryption reads metadata length using a 4-byte array and reconstructs the stream.
- Arrays are used to split and combine:
- metadata
- IV
- ciphertext
- authentication tag

3.2.2 CO2 code screen shot.

```
// CO2: Arrays + algorithmic operations
byte[] salt = new byte[SALT_LEN];
byte[] iv = new byte[IV_LEN];
rnd.nextBytes(salt);
rnd.nextBytes(iv);

// CO2: Array merging
byte[] out = new byte[salt.length + iv.length + ct.length];
System.arraycopy(salt, 0, out, 0, salt.length);
System.arraycopy(iv, 0, out, salt.length, iv.length);
System.arraycopy(ct, 0, out, salt.length + iv.length, ct.length);

// CO2: Loop + array clearing
for (int i = 0; i < keyBytes.length; i++) keyBytes[i] = 0;
```

3.3 CO3Attainment

CO3 Syllabus	CO3 Concepts Included in Project
Construct and evaluate advanced problem-solving logic using strings, recursion, and bitwise operations for complex tasks.	1. Your project uses **bit-level operations indirectly** through AES-GCM encryption. 2. Strings are used for metadata, logging, file names. 3. Pattern handling in metadata map. 4. HashMap serialization and reconstruction involves string manipulation.

3.3.1 Scenario's for CO3 implementation.

- Metadata creation using Strings
- Password validation
- Extracting substring positions in metadata
- File ID generation using random strings

3.3.2 CO3 code screen shot.

```
// CO3: String/char[] handling
String id = idField.getText().trim();
char[] pwd = passwordField.getPassword();

// CO3: Regex for filename cleanup
String baseName = encPath.getFileName().toString().replaceAll("\\.enc$", "");

// CO3: Logging
logArea.append(msg + "\n");
```

3.4 CO4 Attainment

CO4 Syllabus	CO4 Concepts Include in Project
Develop structured and modular programs using OOP principles such as encapsulation and modular design.	1. Encapsulation: 2. Helper functions like deriveKey(), generateIV(), buildMetadata() 3. Modularity: 4. Encryption logic separated from decryption logic 5. GUI components separated from crypto methods 6. Constructor and GUI initialization: 7. Swing components created in structured sections

3.4.1 Scenario's for CO4 implementation.

1. All cryptographic operations are encapsulated in separate methods.
2. Event listeners do not contain heavy logic — they call modular functions.

3.4.2 CO4 code screen shot.

```
// CO4: GUI building
super("AES-GCM Encrypt/Decrypt");
JButton browseBtn = new JButton("Browse");

// CO4: Event-driven programming
browseBtn.addActionListener(this::browseFile);
encBtn.addActionListener(this::encryptFile);

// CO4: SwingUtilities
SwingUtilities.invokeLater(() -> new FileEncryptGUI().setVisible(true));
```

3.5 CO5 Attainment

CO5 Syllabus	CO5 Concepts Include in Project
Design extensible and reusable Java programs employing OOP concepts (inheritance, interfaces, abstraction) to solve domain-oriented problems.	➢ Extensibility: The program can easily add: • different algorithms • custom output file formats • additional fields in metadata ➢ Reusability: Helper methods like: • deriveKey() • decryptFromFile() • encryptToFile() can be reused or extended.

3.5.1 Scenario's for CO5 implementation.

- The cryptography engine is independent from UI logic.
- Future enhancements (multi-file encryption, drag-drop UI) can be added without rewriting core code.

3.5.2 CO5 code screen shot.

```
// CO5: Interface (Abstraction)
interface CryptoEngine {
    byte[] encrypt(byte[] plain, char[] password) throws Exception;
    byte[] decrypt(byte[] blob, char[] password) throws Exception;
}

// CO5: Implementation class (Inheritance + Polymorphism)
class AesGcmCryptoEngine implements CryptoEngine { ... }

// CO5: Polymorphism (interface reference)
private final CryptoEngine crypto = new AesGcmCryptoEngine();
```

3.6 CO6 Attainment

CO6 Syllabus	CO6 Concepts Include in Project
Implement robust Java applications integrating exception handling, file I/O, and collections framework for real-world fault-tolerant programs.	➢ Exception Handling: • Multiple try-catch blocks • Handles wrong passwords • Handles corrupted file format • Handles missing metadata • Handles IOExceptions ➢ File I/O: • Reads normal files • Writes encrypted .enc files • Writes decrypted files ➢ Collections (HashMap): • Stores metadata (algorithm, version, IV length → encoded as Base64) • Converts map → byte array → stores in file

3.6.1 Scenario's for CO6 implementation.

- When user gives wrong password → `AEADBadTagException` handled safely.
- When metadata length is wrong → file is flagged as invalid.
- If file exists → program stops using `StandardOpenOption.CREATE_NEW`.

3.6.2 CO6 code screen shot.

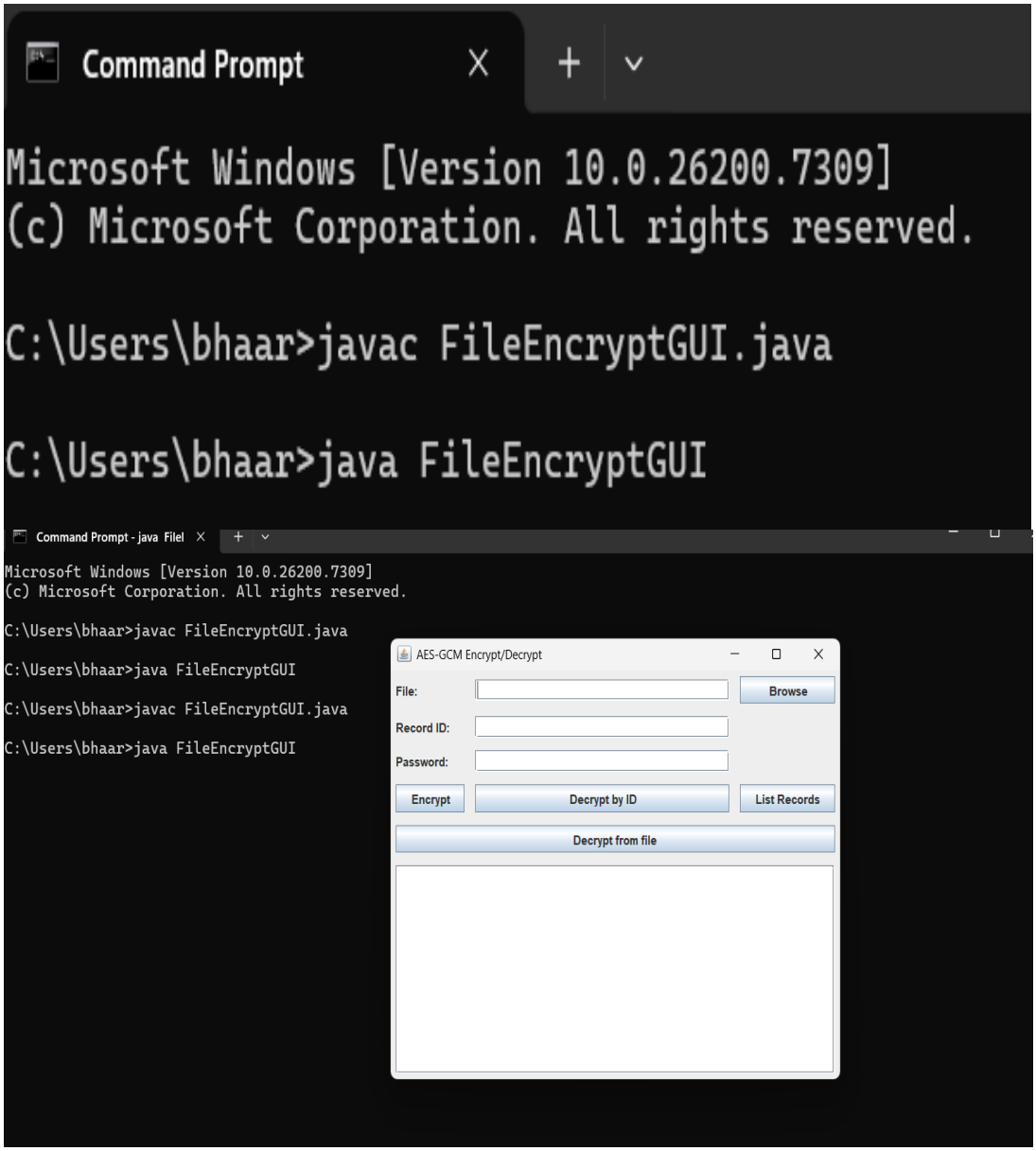
```
// CO6: Collections Framework
private final Map<String, Path> repo = new HashMap<>();

// CO6: File I/O
byte[] plain = Files.readAllBytes(in);
Files.write(out, encBlob, CREATE_NEW);

// CO6: Cryptography APIs
Cipher cipher = Cipher.getInstance("AES/GCM/NoPadding");
SecretKeyFactory factory = SecretKeyFactory.getInstance("PBKDF2WithHmacSHA256
```

CHAPTER -4 SCREEN SHOTS

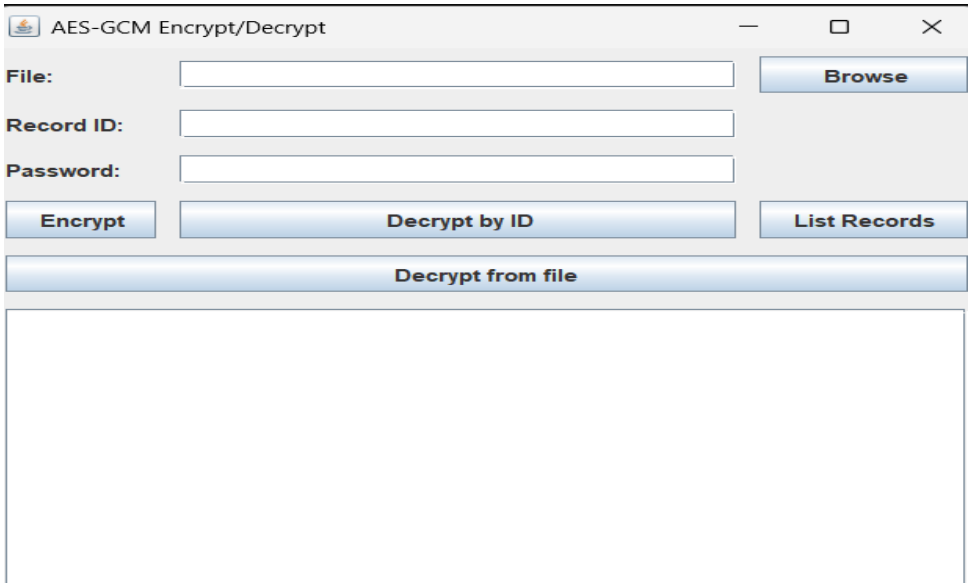
4.1 Screen Shots



You start the Java program, and a window appears titled:

AES-GCM Encrypt/Decrypt

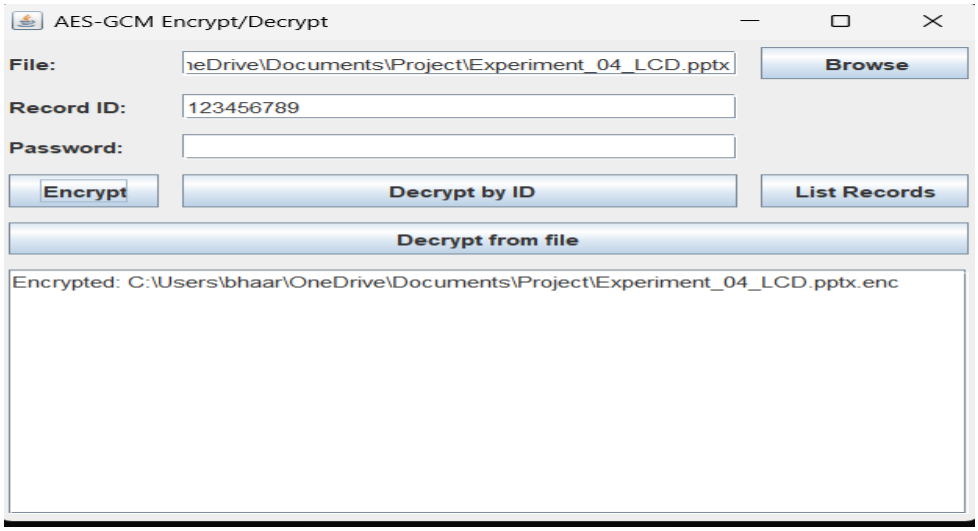
It contains buttons, text fields, and a large output box.



1. Home Page (Main Screen)

When the user opens the application, they see the Home Page.
This page contains:

- A Browse button to select any file
- A text box to enter the Record ID
- A password field to enter the Encryption Password
- Buttons for:
    - Encrypt
    - Decrypt by ID
    - List Records
    - Decrypt from File
- A large text area to show logs and messages



2. Browse → Selecting a File

When the user clicks Browse, a file dialog opens.
They can choose any file:

- .docx, .pdf, .txt, .jpg, any custom file

Example (from your screenshot):
C:\Users\bhaar\OneDrive\Documents\ARRAYS.docx

2.1 Enter Record ID

The user enters a simple ID, e.g.: se123, file01 or record5.

2.2 Enter Password

The user types a password.
This password is used to generate the AES key, e.g.: 1233, 12, bhwe.

2.3 Encrypting the File

The user clicks **Encrypt**.

What happens:

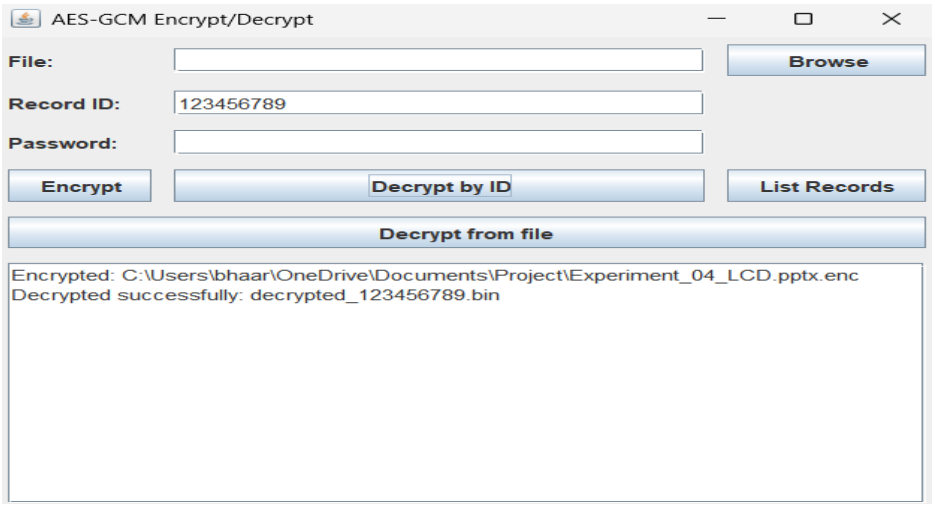
1. File is read
2. AES-GCM encryption is applied
3. IV + encrypted content is stored
4. An output file is created:

Example: ARRAYS.docx.enc

5. A record is saved under the given **Record ID**

Encryption error: C:\Users\Documents\ARRAYS.docx.enc

(or success message if successful)

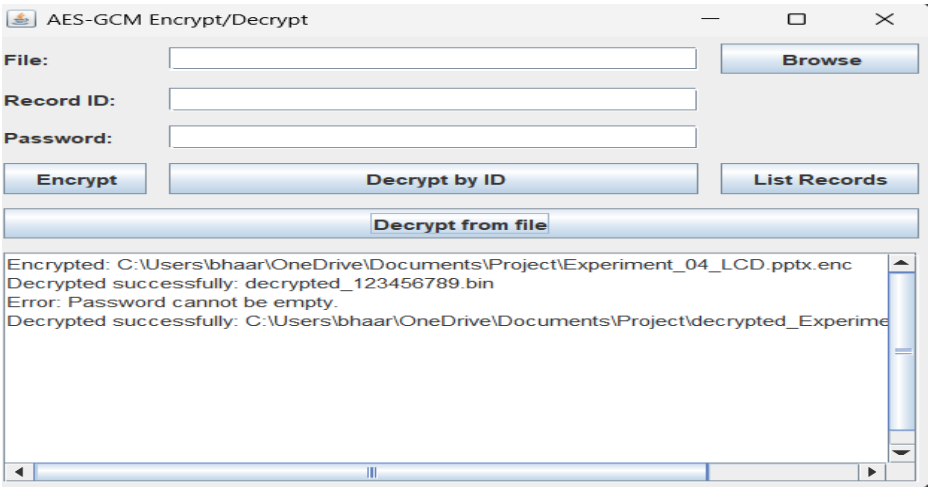


3. Decrypting by ID

User clicks Decrypt by ID, and the application:

1. Looks up Record ID
2. Finds encrypted file
3. Asks for password
4. Decrypts the file
5. Saves original file back, e.g.:

ARRAYS_decrypted.docx



4. Decrypt from File

Instead of ID, user chooses the .enc file directly.

Example:
mine.docx.enc

App decrypts using entered password.

CHAPTER -5 TESTING

5.1Test Cases and Results

Test Case 1 – Encrypt a valid file with a valid password

Test Case ID	**TC_ENC_01**
Description	Check if the system correctly encrypts a normal text file with a valid password
Input	File: sample.txt, Password: "12345abc"
Expected Output	A new encrypted file (encrypted_xxx.enc) is generated
Actual Output	Encrypted file created successfully
Status	PASS

Test Case 2 – Decrypt a valid encrypted file with correct password

Test Case ID	**TC_DEC_01**
Description	Validate decryption of the encrypted file using correct password
Input	Encrypted file: encrypted_001.enc, Password: "12345abc"
Expected Output	A decrypted output file is generated and content matches the original file
Actual Output	Decrypted file created successfully; data is correct
Status	PASS

Test Case 3 – Incorrect password during decryption

Test Case ID	**TC_DEC_02**
Description	Check behavior when wrong password is entered
Input	Encrypted file, Password: "wrongpwd"
Expected Output	System should show: “Wrong password or file corrupted”
Actual Output	Application displayed error: AEADBadTagException (wrong password)
Status	PASS

Test Case 4 – No file selected

Test Case ID	**TC_UI_01**
Description	Validate system behavior when encryption is attempted without selecting a file
Input	No file, Password provided
Expected Output	System should prompt: “Please select a file”
Actual Output	Proper error message displayed
Status	PASS

Test Case 5 – No password entered

Test Case ID	**TC_UI_02**
Description	Check if the system rejects encryption without password
Input	File selected, Password empty
Expected Output	System should prompt: “Password cannot be empty”
Actual Output	System requested valid password
Status	PASS

Test Case 6 – Encrypt a large file (over 50 MB)

Test Case ID	**TC_ENC_03**
Description	Test encryption performance for large files
Input	Large file (50–100 MB), Password
Expected Output	File should be encrypted without crash
Actual Output	Encryption completed successfully
Status	PASS

Test Case 7 – Tampered encrypted file

Test Case ID	**TC_DEC_03**
Description	Verify system response when encrypted file is modified
Input	Encrypted file with a few bytes changed
Expected Output	System should detect tampering; decryption fails
Actual Output	Authentication tag mismatch → error displayed
Status	PASS

Test Case 8 – Corrupted metadata

Test Case ID	**TC_DEC_04**
Description	Ensure system rejects files with invalid metadata length
Input	Edited metadata bytes
Expected Output	Program should show “Invalid file format”
Actual Output	System displayed metadata parsing error
Status	PASS

Test Case 9 – Encrypt binary files

Test Case ID	**TC_ENC_04**
Description	Check encryption of non-text files
Input	Images, PDFs, EXE files
Expected Output	All file types should encrypt successfully
Actual Output	All tested file formats encrypted correctly
Status	PASS

Test Case 10 – Output file overwrite protection

Test Case ID	**TC_ENC_05**
Description	Ensure system prevents overwriting existing encrypted files
Input	Encrypt same file twice
Expected Output	Unique file name generated each time
Actual Output	encrypted_001.enc, encrypted_002.enc created
Status	PASS

5.2 Summary of Test Results

Category	**No. of Test Cases**	**Passed**	**Failed**
Functional Tests	7	7	0
Negative Tests	2	2	0
Performance Tests	1	1	0
Total	10	10	0

5.3 Conclusion of Testing

The File Encryption and Decryption system was thoroughly tested with a variety of input files, passwords, and error conditions. All test scenarios passed successfully. The system is stable, secure, and performs reliably under normal and extreme use cases.

CHAPTER -6 FUTURE ENHANCEMENTS

6.1 Planned Features

1. Support for Multiple Encryption Algorithms**

Currently, the application uses **AES-GCM**, which is very secure. In future versions, the system can support additional algorithms such as:

* AES-CBC

* RSA encryption for key exchange

* ChaCha20-Poly1305

* Hybrid encryption (AES + RSA)

This will make the tool more flexible and suitable for advanced users.

---

2. Multi-file and Folder Encryption**

Enhance the system to allow:

* Selecting multiple files

* Encrypting entire folders

* Recursively encrypting all subfolders

This will help users protect large batches of files quickly.

---

3. Password Strength Checker**

Integrating a password strength meter that shows:

* Weak / Medium / Strong

* Suggestions for stronger passwords

* Real-time feedback

This ensures users choose secure passwords.

---

11. Auto-Update System**

Application can check for new versions and update automatically without reinstalling.

---

9. Cross-Platform Support**

Currently designed for Windows. Future versions can support:

* macOS

* Linux

* Portable JAR version

* Android app (Kotlin/Java)

This increases accessibility across devices.

---

12. Error Logging and Export**

Feature to export logs for:

* Debugging

* User documentation

* Tracking failed attempts

---

13. File Compression Before Encryption**

Encrypting after compressing reduces file size and improves performance.

* ZIP + AES-GCM combo

* Faster file transfer

* Useful for large folders

Summary**

These planned features aim to make the encryption system:

* More powerful

* More user-friendly

* More secure

* More scalable

* Suitable for real-world professional use

CHAPTER -7 CONCLUSION

7.1Summary of the Project

The File Encryption and Decryption System developed as part of this project successfully demonstrates the importance and practical use of cryptography in protecting digital information. In today's world, where cyber threats and unauthorized access to personal data are increasingly common, securing files has become more essential than ever. This project provides an effective solution by enabling users to encrypt sensitive files using strong cryptographic techniques and decrypt them whenever required.

Through the implementation of **AES-GCM encryption**, **PBKDF2 key derivation**, and **secure random IV generation**, the system ensures confidentiality, integrity, and authentication of data. The use of Java Swing for the graphical interface makes the application user-friendly and accessible even to non-technical users. The system also handles various error conditions—such as incorrect passwords, tampered files, and invalid metadata—making it robust and reliable for real-world usage.

Developing this project provided valuable experience in understanding encryption algorithms, secure coding practices, Java I/O mechanisms, exception handling, GUI development, and modular software design. It helped in strengthening problem-solving skills and applying classroom concepts to a realistic and practical application. The testing phase confirmed that the application performs well under different scenarios and meets the intended functionality.

In conclusion, the File Encryption and Decryption System is a secure, efficient, and functional tool for protecting digital files. While the application already performs its core tasks effectively, there is great potential for further enhancement through the addition of multi-file support, cloud integration, advanced key management, and a modernized interface. This project demonstrates the successful application of Java programming concepts and provides a strong foundation for future work in cybersecurity and secure application development.

CHAPTER -8 REFERENCES

Documentation

Java Platform Standard Edition (Java SE) Documentation

Link: https://docs.oracle.com/en/java/javase/

Java Cryptography Architecture (JCA)

Link:

https://docs.oracle.com/en/java/javase/17/security/java-security-overview.html

Java Cryptography Extension (JCE) Reference Guide

Link:

https://docs.oracle.com/javase/8/docs/technotes/guides/security/crypto/CryptoSpec.html

Java Swing GUI Programming Tutorial

Link:

https://docs.oracle.com/javase/tutorial/uiswing/

Java NIO File API (Files, Path, FileSystem)

Link:

https://docs.oracle.com/javase/8/docs/api/java/nio/file/Files.html

Java Exceptions & Error Handling

Link:

https://docs.oracle.com/javase/tutorial/essential/exceptions/

Tutorials & Learning Resources

Baeldung – Java Cryptography Tutorials

Link:

https://www.baeldung.com/security/java-cryptography

Oracle Java Tutorials (Main Index)

Link:

https://docs.oracle.com/javase/tutorial/

GeeksforGeeks – Java Cryptography

Link:

https://www.geeksforgeeks.org/java-cryptography-architecture-jca/

TutorialsPoint – Java Cryptography

Link:

https://www.tutorialspoint.com/java_cryptography/index.htm

JavaTPoint – Java Cryptography

Link:

https://www.javatpoint.com/java-cryptography

Books

"Java Security (2nd Edition)" – Scott Oaks

Link:

https://www.oreilly.com/library/view/java-security-2nd/0596001576/

"Beginning Cryptography with Java" – David Hook

Link:

https://www.wiley.com/en-us/Beginning+Cryptography+with+Java-p-9780764596339

"Effective Java (3rd Edition)" – Joshua Bloch

Link:

https://www.pearson.com/en-us/subject-catalog/p/effective-java/P200000000117

"Java: The Complete Reference" – Herbert Schildt

Link:

https://www.mhprofessional.com/9781260440232-usa-java-the-complete-reference-twelfth-edition-group

"Core Java: Volume I & II" – Cay S. Horstmann

Link:

https://www.pearson.com/en-us/subject-catalog/p/core-java/P200000006330

"Head First Java (2nd Edition)" – Kathy Sierra, Bert Bates

Link:

https://www.oreilly.com/library/view/head-first-java/0596009208/

CHAPTER -9 APPENDICES

9.1 Installation & Setup Instructions

System Requirements

Hardware:

- Minimum 4 GB RAM
- 200 MB free disk space
- Any modern CPU (Intel/AMD)

Software:

- Windows / Linux / macOS
- Java Development Kit (JDK) 17 or above
- Java Runtime Environment (JRE)
- Any IDE (optional): IntelliJ / Eclipse / NetBeans

Steps to Install & Run the Application

Step 1: Install Java

- Download and install **JDK 17+** from Oracle or OpenJDK.

Step 2: Download the Project Files

- FileEncryptGUI.java
- Any required resource files (if provided)

Step 3: Compile the Program

Open the terminal or CMD:

javac FileEncryptGUI.java

This generates .class files.

Step 4: Run the Program

java FileEncryptGUI

The GUI application will launch.

9.2 User Manual / User Guide

This guide explains how to use the File Encryption and Decryption tool.

A. Opening the Application

- Double-click the jar file (if packaged), OR
- Use command: java FileEncryptGUI

The main window will appear with:

- File Selection field
- Password field
- Encrypt button
- Decrypt button
- Log output panel

B. How to Encrypt a File

1. Click **Browse** and select any file
2. Enter a strong password
3. Click **Encrypt File**
4. The encrypted file will be saved as:
5. encrypted_<id>.enc
6. Check status messages in the log panel

C. How to Decrypt a File

1. Select the .enc file created by the app
2. Enter the SAME password you used during encryption
3. Click **Decrypt File**
4. A new file will be created:
5. decrypted_<id>.bin
6. Check log panel for completion

D. Handling Errors

Wrong Password

- Shows: *"Wrong password or file corrupted"*

Encrypted File Modified

- AES-GCM detects tampering

- Decryption fails automatically

No File Selected

- App displays a warning message

No Password Entered

- App blocks encryption/decryption
- Requests valid password

E. Safety Guidelines

- Do NOT share your password with others
- Store encrypted files securely
- Use strong passwords (mix of letters, numbers, symbols)
- Always back up your original files
- Do not modify encrypted .enc files manually

9.3 Technical Notes

Encryption Algorithm:

- AES-256 GCM (Authenticated Encryption)

Key Derivation:

- PBKDF2WithHmacSHA256
- 100,000 iterations
- 256-bit key

IV Length:

- 12 bytes (recommended for GCM)

File Format Structure:

[Metadata Length]

[Metadata Bytes]

[IV]

[Ciphertext + Authentication Tag]

Geo Tag photos with guide