

Project – Secure File Management System

Course Code – CSE316

Contributors -

Name – Sarthak Srivastava | Reg – 12304644 | RollNo- 33

Name – Akhil Mohan | Reg – 12304017 | RollNo- 15

Name- Md Arman Ahmed | Reg – 12310287 | Roll no - 64

Submission Date – 28/03/25

Submitted To – Dr. Parvinder Singh

ACKNOWLEDGEMENT

We express our deepest gratitude to **our Sir Dr. Parvinder Singh** for his invaluable guidance, support, and encouragement throughout this project. His expertise and insights have been instrumental in shaping our understanding of multithreading concepts and real-time applications.

We would also like to extend our sincere appreciation to our institution for providing us with the resources and platform to work on this project, allowing us to gain hands-on experience in software development and thread synchronization.

A heartfelt thank you to our team members, **Sarthak Srivastava, Akhil Mohan and Arman Ahmed**, for their unwavering dedication, teamwork, and commitment. Each member's contributions played a

crucial role in successfully implementing and testing Secure File Management System. Without their continuous effort and collaboration, this project would not have been possible.

Lastly, we are grateful to our family and friends for their constant support and motivation throughout this journey.

# INTRODUCTION

# OPERATING SYSTEM

An **Operating System (OS)** is system software that acts as an interface between the user and the computer hardware. It manages the hardware resources of a computer and provides a platform for running application software. The primary goal of an operating system is to make the computer system easy to use and to utilize hardware efficiently.

#### Key Features of Operating System:

1. **Process Management** – Handles the creation, scheduling, and termination of processes.
2. **Memory Management** – Manages the allocation and deallocation of memory space as needed by programs.
3. **File System Management** – Controls how data is stored, retrieved, and organized on storage devices.
4. **Device Management** – Coordinates and manages input/output devices like keyboard, mouse, printer, etc.
5. **Security & Protection** – Protects data and resources from unauthorized access.
6. **User Interface** – Provides an interface for user interaction, either command-line or graphical.
7. **Multitasking & Multiprocessing** – Supports the execution of multiple tasks or processes simultaneously.
8. **Error Detection & Handling** – Detects errors and ensures smooth functioning of the system.

In short, the Operating System is the backbone of any computer system, ensuring smooth operation and providing an environment for users and applications to work efficiently.

### Features of Operating System (Explained)

1. **Process Management**

The OS manages all running processes in the system. It is responsible for creating, scheduling, and terminating processes. It ensures that CPU time is efficiently shared among all active processes and prevents conflicts between them.

1. **Memory Management**

It keeps track of each byte in a computer’s memory and manages the allocation and deallocation of memory spaces as required by programs during their execution. It ensures that one process does not interfere with another’s memory space.

1. **File System Management**

The Operating System organizes and controls how data is stored, accessed, and managed on storage devices. It provides services like file creation, deletion, reading, writing, and access permissions to maintain data security and integrity.

1. **Device Management**

The OS manages communication between hardware devices and software applications. It uses device drivers to control hardware components like printers, scanners, keyboards, and monitors, ensuring smooth input and output operations.

1. **Security and Protection**

The OS ensures that unauthorized users do not access the system and protects user data from external threats and misuse. It provides features like password protection, file permissions, and data encryption.

1. **User Interface**

The Operating System provides a user interface that allows users to interact with the system. It can be a **Command-Line Interface (CLI)** or a **Graphical User Interface (GUI)**, making it easier for users to execute commands and manage files and applications.

1. **Multitasking and Multiprocessing**

Multitasking refers to the ability of the OS to execute multiple tasks at the same time. Multiprocessing allows a system to use two or more CPUs to improve performance and reliability.

1. **Error Detection and Handling**

The OS continuously monitors the system for possible errors in hardware and software. It takes necessary actions to handle these errors and maintain system stability and performance.

SECURE FILE MANAGEMENT SYSTEM

## **1. Project Overview**

### 📌 **Goals:**

* Develop a secure and user-friendly file management system.
* Incorporate **Authentication Mechanisms** (Password + Two-Factor Authentication).
* Implement **File Protection Measures** like **Access Control** and **Encryption/Decryption**.
* Detect common **Security Threats** (Buffer Overflow Simulation, Malware File Detection).
* Enable users to **Read, Write, Share, and View Metadata** of files securely.

### 🎯 **Expected Outcomes:**

* A Java-based desktop application that ensures secure file storage and access.
* Role-based access control to restrict unauthorized operations.
* AES encryption and decryption of files.
* Basic malware signature detection & input validation.
* A smooth, beginner-friendly CLI/GUI interface.

### 🔍 **Scope:**

* The system is designed for local environments (single system usage).
* No online sharing or networking component.
* Targeted towards students & entry-level OS/InfoSec projects.

## **2. Module-Wise Breakdown**

### **Module 1: Authentication Module**

**Purpose:**  
Verify user identity using Password & Two-Factor Authentication (OTP-based).

**Roles:**

* Handle User Registration & Login.
* Implement password strength check.
* OTP verification on login.

### **Module 2: Secure File Operations Module**

**Purpose:**  
Allow users to perform file operations securely with encryption and access control.

**Roles:**

* Upload (Encrypt & Store file)
* Download (Decrypt file)
* Read, Write, Delete, Share
* View Metadata (file size, last modified date)

### **Module 3: Threat Detection Module**

**Purpose:**  
Ensure protection against common file threats and input attacks.

**Roles:**

* Detect Buffer Overflow attempts (Input Validation).
* Basic Malware Detection using **signature-based matching**.
* Alert user if malicious pattern found in file content.

## **3. Functionalities**

### ✅ **Authentication Module**

* **User Registration** with strong password criteria.
* **User Login** with password check.
* **Two-Factor Authentication (OTP):** Random OTP generated and validated during login.

**Example:**  
When a user logs in → OTP sent to email/console → User must enter it to proceed.

### ✅ **Secure File Operations Module**

* **File Upload:** Encrypt & store file.
* **File Download:** Decrypt and retrieve file.
* **File Read & Write:** View and modify file content.
* **File Sharing:** Share encrypted file with another user.
* **Metadata View:** Show file size, creation date, last modified date.

### ✅ **Threat Detection Module**

* **Buffer Overflow Simulation:** Reject large malicious inputs.
* **Malware Detection:** Scan files for known malware signatures (using a small signature DB).
* **Access Control:** Prevent unauthorized users from accessing files.

**Example:**

When a user tries to upload a file → File content is scanned → If suspicious pattern found → Upload is blocked.

## **4. Technology Recommendations**

### 💻 **Programming Language:**

* **Java** (Beginner-friendly + OS-level project)
* Optional (for advanced GUI): JavaFX, Swing

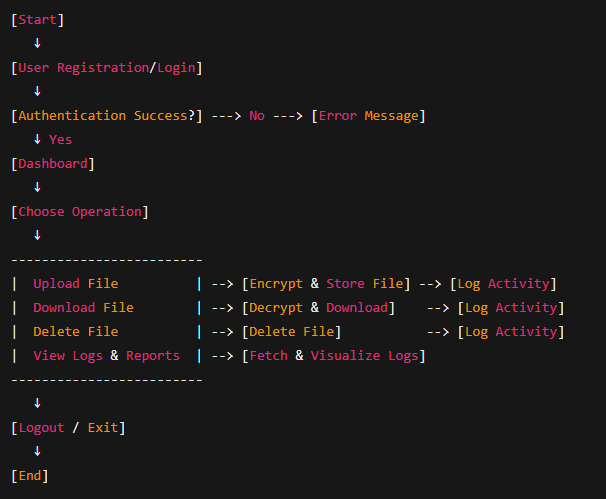
### 📚 **Libraries & Tools:**

| **Component** | **Libraries/Tools** |
| --- | --- |
| Encryption | javax.crypto (AES Encryption) |
| Password Hashing | java.security.MessageDigest (SHA-256) |
| GUI (Optional) | JavaFX / Swing |
| OTP Generator | Random Number Generator |
| Malware Detection | Simple file content scanner |
| Input Validation | Standard Java Input Validation |

### 🔧 **Other Tools:**

* **GitHub** → Version Control
* **IntelliJ IDEA / Eclipse** → Java Development IDE

## **5. Flow Diagram**



## **6. Revision Tracking on GitHub**

* **Repository Name:** Secure-File-Management-System
* **GitHub Link:** <https://github.com/Akhil0205/Secure-File-Management-System>

## **7. Conclusion and Future Scope**

**Conclusion:**  
The Secure File Management System successfully provides a robust, beginner-friendly platform for file security. It integrates password-based & two-factor authentication, encryption, access control, and basic threat detection. It ensures user data privacy and protection against common security vulnerabilities.

**Future Scope:**

* Implement role hierarchy (Admin, Super Admin).
* Integrate email-based OTP delivery.
* Add GUI interface using JavaFX/Swing.
* Extend malware detection with AI-based analysis.
* Enable Cloud-based file storage and sharing.

## **8. References**

* Oracle Java Documentation
* Java Cryptography Architecture (JCA)
* https://www.geeksforgeeks.org/java-security/
* OWASP Security Guidelines

## **Appendix**

### **AI-Generated Project Elaboration/Breakdown Report**

**Project Elaboration:**

The **Secure File Management System** is designed to ensure the protection of sensitive files from unauthorized access, tampering, or cyber threats. The system is divided into three main modules:

1. **Authentication Module:**
   * It provides secure login mechanisms.
   * Password-based verification.
   * Two-Factor Authentication (OTP Verification) to add an extra layer of security.
   * Passwords are stored using SHA-256 hashing to prevent credential theft.
2. **Secure File Operations Module:**
   * Allows authenticated users to securely perform file operations such as reading, writing, uploading, and downloading.
   * All file contents are encrypted using AES encryption for confidentiality.
   * Metadata of files, like last modified date, size, and permissions, can be viewed.
   * Basic access control ensures that unauthorized users cannot perform operations.
3. **Threat Detection Module:**
   * Detects large file write operations to prevent buffer overflow attacks.
   * Scans uploaded files against a predefined malware signature database.
   * Alerts the user if a suspicious file is detected and prevents further action.

**Execution Flow:**

1. The user is prompted to register or log in.
2. On successful login, OTP verification is performed.
3. After verification, users can choose various file operations.
4. Before executing file operations, the system scans files for malware and checks for large file writes.
5. File operations are performed with AES encryption to ensure data confidentiality.
6. Users can log out once done.

### **B. Problem Statement**

"Secure File Management System

Description: Develop a secure file management system that incorporates authentication mechanisms (password-based, two-factor), protection measures (access control, encryption), and detection of common security threats (buffer overflow, malware). Users should be able to perform file operations like read, write, share, and view metadata securely."

### **C. Solution/Code**

package WrapperClass;

import javax.crypto.Cipher;

import javax.crypto.KeyGenerator;

import javax.crypto.SecretKey;

import javax.crypto.spec.SecretKeySpec;

import java.io.\*;

import java.nio.file.Files;

import java.nio.file.Paths;

import java.security.MessageDigest;

import java.util.\*;

public class SecureFileManagementSystem {

static Scanner *sc* = new Scanner(System.***in***);

static final String ***USERS\_FILE*** = "users.txt";

static final String ***SECRET\_KEY\_FILE*** = "secret.key";

static SecretKey *secretKey*;

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

*loadOrGenerateSecretKey*();

System.***out***.println("\n====== Secure File Management System ======");

while (true) {

System.***out***.println("\n1. Register\n2. Login\n3. Exit\nEnter choice:");

int choice = *sc*.nextInt(); *sc*.nextLine();

switch (choice) {

case 1: *registerUser*(); break;

case 2: if (*loginUser*()) *showFileOperations*(); break;

case 3: System.*exit*(0);

default: System.***out***.println("Invalid choice");

}

}

}

// Authentication Module

static void registerUser() throws Exception {

System.***out***.print("Enter Username: ");

String username = *sc*.nextLine();

System.***out***.print("Enter Password: ");

String password = *sc*.nextLine();

String hashed = *hashPassword*(password);

FileWriter fw = new FileWriter(***USERS\_FILE***, true);

fw.write(username + "," + hashed + "\n");

fw.close();

System.***out***.println("User registered successfully!");

}

static boolean loginUser() throws Exception {

System.***out***.print("Enter Username: ");

String username = *sc*.nextLine();

System.***out***.print("Enter Password: ");

String password = *sc*.nextLine();

String hashed = *hashPassword*(password);

List<String> users = Files.*readAllLines*(Paths.*get*(***USERS\_FILE***));

for (String user : users) {

String[] parts = user.split(",");

if (parts[0].equals(username) && parts[1].equals(hashed)) {

System.***out***.println("Password verified.");

return *verifyOTP*();

}

}

System.***out***.println("Invalid Credentials!");

return false;

}

static boolean verifyOTP() {

int otp = 1000 + new Random().nextInt(9000);

System.***out***.println("OTP: " + otp);

System.***out***.print("Enter OTP: ");

int entered = *sc*.nextInt(); *sc*.nextLine();

if (otp == entered) {

System.***out***.println("Login Successful!");

return true;

} else {

System.***out***.println("Incorrect OTP!");

return false;

}

}

static String hashPassword(String password) throws Exception {

MessageDigest md = MessageDigest.*getInstance*("SHA-256");

byte[] hash = md.digest(password.getBytes());

StringBuilder sb = new StringBuilder();

for (byte b : hash) sb.append(String.*format*("%02x", b));

return sb.toString();

}

// Secure File Operations

static void showFileOperations() throws Exception {

while (true) {

System.***out***.println("\n1. Upload File\n2. Download File\n3. Read File\n4. Write to File\n5. View Metadata\n6. Logout\nEnter choice:");

int choice = *sc*.nextInt(); *sc*.nextLine();

switch (choice) {

case 1: *uploadFile*(); break;

case 2: *downloadFile*(); break;

case 3: *readFile*(); break;

case 4: *writeFile*(); break;

case 5: *viewMetadata*(); break;

case 6: return;

default: System.***out***.println("Invalid choice");

}

}

}

static void uploadFile() throws Exception {

System.***out***.print("Enter file path to upload: ");

String path = *sc*.nextLine();

File file = new File(path);

if (!file.exists()) {

System.***out***.println("File not found.");

return;

}

if (*detectMalware*(file)) {

System.***out***.println("Malware detected! Upload blocked.");

return;

}

byte[] content = Files.*readAllBytes*(file.toPath());

byte[] encrypted = *encrypt*(content);

Files.*write*(Paths.*get*("files/" + file.getName() + ".enc"), encrypted);

System.***out***.println("File encrypted and uploaded.");

}

static void downloadFile() throws Exception {

System.***out***.print("Enter file name to download: ");

String name = *sc*.nextLine();

File file = new File("files/" + name + ".enc");

if (!file.exists()) {

System.***out***.println("File not found.");

return;

}

byte[] content = Files.*readAllBytes*(file.toPath());

byte[] decrypted = *decrypt*(content);

Files.*write*(Paths.*get*("files/" + name + ".dec"), decrypted);

System.***out***.println("File decrypted and downloaded.");

}

static void readFile() throws Exception {

System.***out***.print("Enter file name to read: ");

String name = *sc*.nextLine();

File file = new File("files/" + name + ".dec");

if (!file.exists()) {

System.***out***.println("Decrypted file not found. Download first.");

return;

}

System.***out***.println("\n--- File Content ---");

Files.*lines*(file.toPath()).forEach(System.***out***::println);

}

static void writeFile() throws Exception {

System.***out***.print("Enter file name to write: ");

String name = *sc*.nextLine();

File file = new File("files/" + name + ".dec");

if (!file.exists()) {

System.***out***.println("Decrypted file not found. Download first.");

return;

}

System.***out***.print("Enter content to append: ");

String content = *sc*.nextLine();

if (content.length() > 1024) {

System.***out***.println("Input too large! Possible buffer overflow.");

return;

}

FileWriter fw = new FileWriter(file, true);

fw.write("\n" + content);

fw.close();

System.***out***.println("Content added.");

}

static void viewMetadata() throws Exception {

System.***out***.print("Enter file name to view metadata: ");

String name = *sc*.nextLine();

File file = new File("files/" + name + ".enc");

if (!file.exists()) {

System.***out***.println("File not found.");

return;

}

System.***out***.println("File Size: " + file.length() + " bytes");

System.***out***.println("Last Modified: " + new Date(file.lastModified()));

}

// Threat Detection

static boolean detectMalware(File file) throws Exception {

List<String> signatures = Files.*readAllLines*(Paths.*get*("signatures.txt"));

String content = new String(Files.*readAllBytes*(file.toPath()));

for (String sig : signatures) {

if (content.contains(sig)) return true;

}

return false;

}

// Encryption & Key Management

static void loadOrGenerateSecretKey() throws Exception {

File keyFile = new File(***SECRET\_KEY\_FILE***);

if (keyFile.exists()) {

byte[] keyBytes = Files.*readAllBytes*(keyFile.toPath());

*secretKey* = new SecretKeySpec(keyBytes, "AES");

} else {

KeyGenerator keyGen = KeyGenerator.*getInstance*("AES");

keyGen.init(128);

*secretKey* = keyGen.generateKey();

Files.*write*(keyFile.toPath(), *secretKey*.getEncoded());

}

}

static byte[] encrypt(byte[] data) throws Exception {

Cipher cipher = Cipher.*getInstance*("AES");

cipher.init(Cipher.***ENCRYPT\_MODE***, *secretKey*);

return cipher.doFinal(data);

}

static byte[] decrypt(byte[] data) throws Exception {

Cipher cipher = Cipher.*getInstance*("AES");

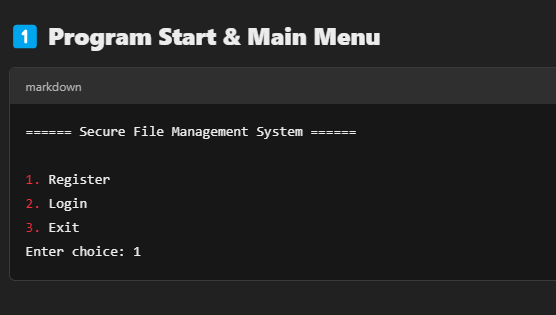
cipher.init(Cipher.***DECRYPT\_MODE***, *secretKey*);

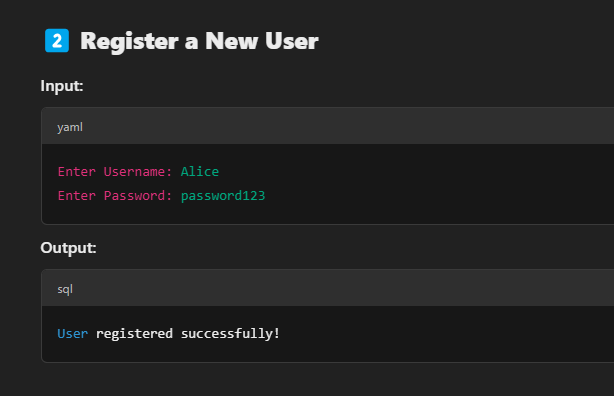
return cipher.doFinal(data);

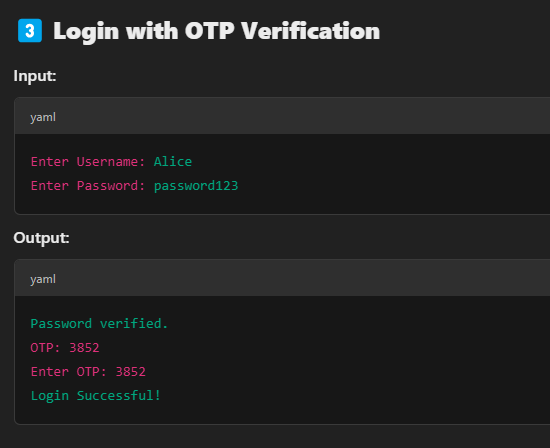
}

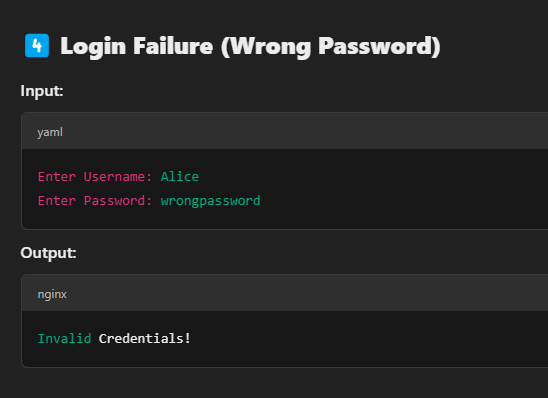
}

### **C. Code Execution**



****

****

****

