Secure File Sharing System Report

Submitted By: Gourav Swaroop

Date: 26/08/2025

About the Task

This project focuses on building a **Secure File Sharing System (SFS)** that allows users to upload and download files safely. The core of the system is **encryption** — ensuring that files are protected during both storage and transfer.

The task simulates real-world client requirements in industries like healthcare, legal, and corporate environments, where **data confidentiality** is critical.

Objectives

- Develop a **web portal** for secure file sharing using Flask.
- Implement **AES encryption** to secure files before storage and decrypt them during retrieval.
- Provide a simple **frontend interface** for uploads and downloads.
- Ensure **secure key handling** and test data integrity.
- Document architecture, code, and security measures.

Tools & Technologies Used

- **Python Flask** – Backend framework for file upload/download
- **PyCryptodome** – AES encryption/decryption library
- **HTML, CSS, JavaScript** – User interface
- **Git & GitHub** – Version control
- **Postman / curl** – API testing

Implementation Overview

1. **Homepage** – Acts as the entry point with navigation links.
2. **Upload Page** – Allows users to upload files securely.

3. **AES Encryption** – Files are encrypted before being saved on the server.
4. **File Storage** – Encrypted files are stored in the /uploads/ folder.
5. **Files List Page** – Displays all encrypted files available for download.
6. **Download & Decryption** – Users can download files, which are decrypted back to their original form.

Screenshots & Evidence

1. Home Page

Shows the landing page of the Secure File Sharing system.

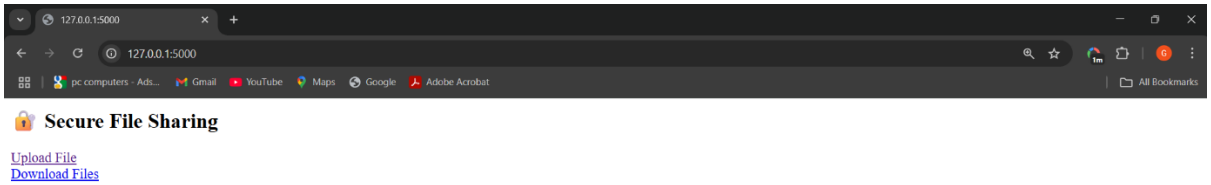

Figure 1: Home Page of the Secure File Sharing System

2. Upload Page

Demonstrates the file upload form where users can select and upload files securely.

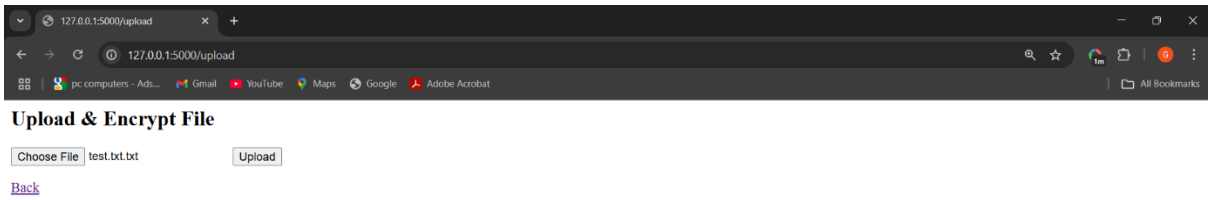

Figure 2: File Upload Page with encryption in action

3. Upload Success

Confirmation message after a file is encrypted and uploaded successfully.

Figure 3: Upload Success Confirmation Message

4. Files List

Displays all uploaded encrypted files available for download.

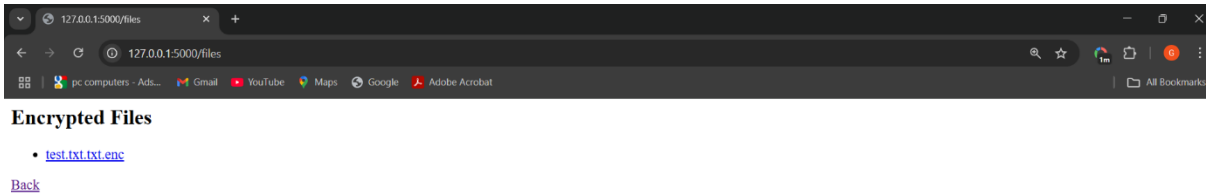

Figure 4: Files List Showing All Encrypted Files Available for Download

5. Flask Server Running

Screenshot of the Flask development server console showing the backend running.

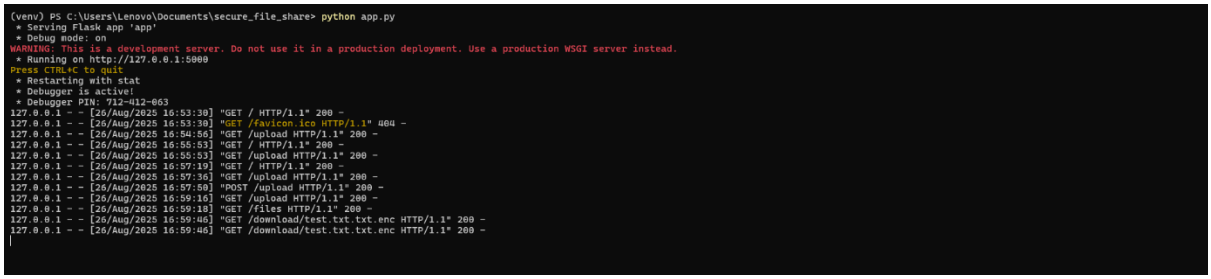

Figure 5: Flask Server Console Running the Backend

6. Download Prompt

Browser/system download popup when retrieving an encrypted file.

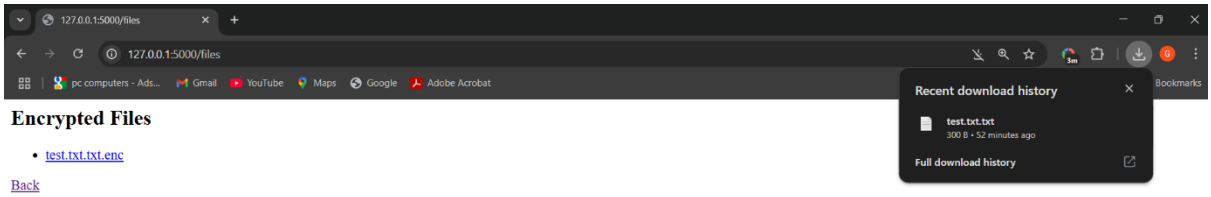

Figure 6: Download Prompt for Encrypted File

7. Encrypted File

Screenshot showing the .enc version of the uploaded file opened in a text editor, proving the content is unreadable.

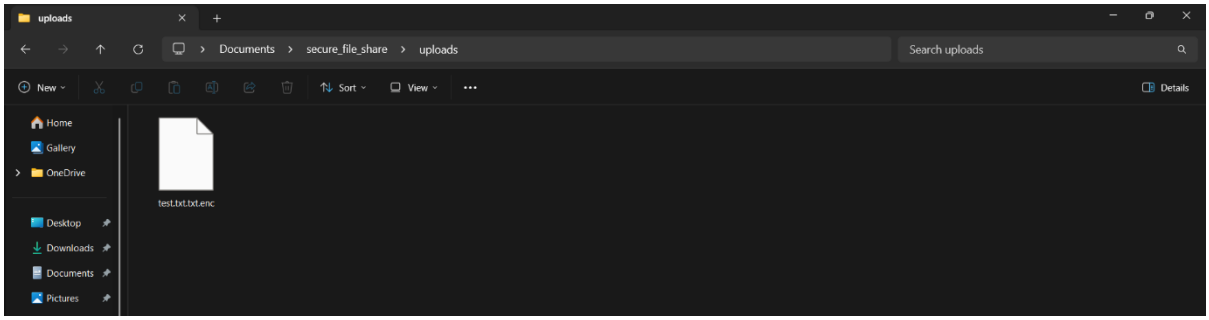

Figure 7: Encrypted File Opened in Text Editor (Unreadable Content)

8. Decrypted File

Screenshot showing the decrypted file opened in Notepad, confirming that the original content is intact and matches the uploaded version.

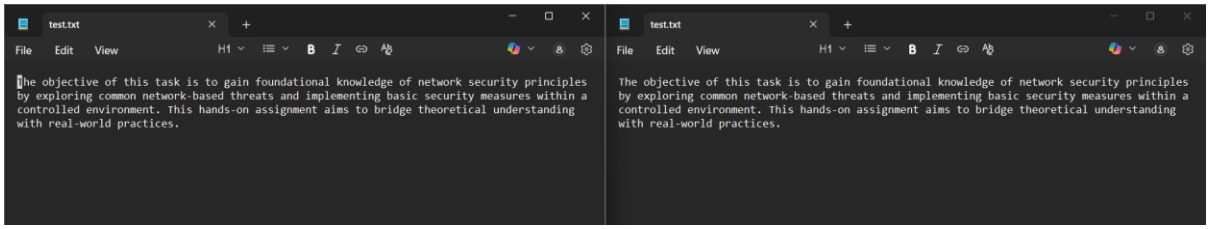

Figure 8: Decrypted File Opened in Notepad (Original Content Intact)

Security Considerations

- **AES Encryption** ensures confidentiality of files.
- **Key Management**: A secret key is used for encryption/decryption, which must be securely stored.
- **Data Integrity**: Verified by comparing original and decrypted files.
- **Access Control**: Only users with decryption keys can access the actual content.

Skills Gained

- Web development (Flask backend + HTML frontend)
- Implementing **AES encryption** for secure storage
- File handling and **data protection best practices**

- Key management in cryptography
- Using **Git/GitHub** for version control
- Testing APIs with Postman/curl

Conclusion

The Secure File Sharing System was successfully implemented with AES encryption, ensuring files remain protected during both storage and transfer. The project demonstrates the importance of secure file handling and provides a functional prototype that simulates real-world secure data sharing.

Future Improvements

- **Authentication & User Accounts**: Add login system to ensure only authorized users can upload/download files.
- **Role-Based Access Control (RBAC)**: Different permissions for admins and regular users.
- **Cloud Storage Integration**: Store encrypted files securely on platforms like AWS S3 or Google Cloud.
- **Key Management Service (KMS)**: Replace static keys with a secure key vault (e.g., AWS KMS, HashiCorp Vault).
- **Logging & Monitoring**: Add audit logs to track file uploads, downloads, and access patterns.