Project Report: Secure File Storage System with AES-256 Encryption

---

1. Introduction

In the modern digital age, storing sensitive files securely has become increasingly important. Personal documents, financial records, and confidential organizational data are often vulnerable to unauthorized access, accidental leaks, and cyberattacks. Traditional local or cloud storage systems frequently lack strong encryption, leaving files exposed to risk.

To address this, the **Secure File Storage System with AES-256 Encryption** was developed. This system allows users to encrypt and decrypt files locally with a password-derived key, ensuring that only authorized users can access the data. Additionally, it maintains file integrity through cryptographic hashing, providing protection against tampering. The system combines strong security, ease of use, and flexible file management through both a command-line and graphical interface.

---

2. Project Objectives

The primary goals of this project were:

- To implement **AES-256 encryption** for securing local files.
- To provide **end-to-end integrity verification** for stored files using SHA-256 hashing.
- To design both a **CLI and GUI** interface, offering flexibility for different user preferences.
- To enable **bulk file processing** for encrypting or decrypting multiple files efficiently.
- To create a user-friendly system that ensures **confidentiality, authenticity, and reliability** of sensitive data.

---

3. Technology Stack

Component	Technology
Programming Language	Python 3.13
GUI Framework	PyQt5

Component	Technology
Encryption	AES-256 (GCM mode)
Hashing	SHA-256 for integrity verification
File Handling	OS & JSON for metadata
Development Tool	Visual Studio Code
Execution Environment	Windows/Linux/macOS

---

4. System Design

4.1 Encryption Workflow

1. The user provides a password, which is used to derive a strong AES-256 key using **PBKDF2 with salt**.
2. Files are encrypted using **AES-GCM**, which provides both confidentiality and authenticity.
3. Metadata including the original file name, initialization vector (IV), hash, and timestamp is stored securely in a .meta file.

4.2 Decryption Workflow

1. Users select the encrypted file and provide the correct password.
2. The AES key is derived again from the password and salt in the metadata.
3. The system verifies the integrity of the file using the stored hash and GCM authentication tag.
4. If verification passes, the file is decrypted and stored in the decrypted_files folder.

4.3 File Organization

- encrypted_files/ → stores encrypted .enc files.
- decrypted_files/ → stores decrypted files.
- Metadata files (.meta) accompany each encrypted file for verification.

---

5. Features

- **AES-256 Encryption:** Strong symmetric encryption for file confidentiality.
- **Integrity Verification:** SHA-256 hashes prevent tampering and corruption.
- **CLI & GUI Interfaces:** Command-line interface for advanced users and PyQt5 GUI for ease of use.
- **Bulk Processing:** Encrypt or decrypt multiple files simultaneously.
- **Drag-and-Drop Support:** Easy selection of multiple files in the GUI.
- **Secure Password Handling:** Passwords are never stored; keys are derived dynamically.

---

6. Implementation Challenges & Solutions

Challenge	Solution
Handling user errors such as wrong passwords	Added proper error handling and integrity verification to prevent incorrect decryption.
Ensuring file integrity after encryption	Implemented SHA-256 hashing and AES-GCM authentication tags.
Managing multiple files in GUI	Added bulk encryption/decryption and drag-and-drop support.
User-friendly GUI design	Built PyQt5 interface with clear instructions and real-time status messages.
Secure metadata management	Stored all required cryptographic data in separate JSON .meta files.

---

7. Results and Achievements

- **Secure Storage:** AES-256 encryption ensures that files cannot be accessed without the correct password.
- **Data Integrity:** GCM authentication and SHA-256 hash guarantee that tampered files are not decrypted.
- **User-Friendly Interfaces:** CLI and GUI allow flexibility in usage for different user types.
- **Bulk Operations:** Users can process multiple files efficiently.

- **Cross-Platform Support:** The system works on Windows, Linux, and macOS.

---

8. Future Enhancements

- **Master Password:** Implement a single master password to manage multiple files efficiently.
- **Encrypted Cloud Storage:** Integrate secure cloud backup while maintaining AES encryption.
- **Multi-User Sharing:** Allow secure file sharing between users using password-based access.
- **Mobile Application:** Extend GUI functionality to mobile platforms for on-the-go file encryption/decryption.
- **Enhanced GUI Features:** Add progress indicators, file previews, and logs for a more polished interface.

---

9. Conclusion

The **Secure File Storage System with AES-256** successfully provides a reliable and user-friendly solution for protecting sensitive files. By combining strong AES encryption with integrity verification and multiple interface options, the system ensures confidentiality, authenticity, and ease of use.

This project demonstrates the practical application of cryptographic principles in securing local file storage and provides a foundation for future enhancements, such as secure cloud storage or multi-user file sharing systems.

Prepared By: Bhagyashree Satpathy
Date: 24th October 2025
Organization: Elevate Labs