🔐 Secure File Storage System with AES Encryption – Project Report

---

Introduction

In the modern digital landscape, data privacy and security are paramount. Sensitive files shared or stored on systems must be protected from unauthorized access and tampering. This project addresses that need by implementing a Secure File Storage System that utilizes **AES-256 encryption**, one of the most secure symmetric encryption standards, to ensure confidentiality and integrity of user files.

Abstract

The project is a robust Python-based tool that enables users to encrypt and decrypt files securely using a password-based AES-256 encryption system. It features a **graphical user interface (GUI)** built with PyQt5 and supports a **command-line interface (CLI)** for advanced users. The system derives keys securely using PBKDF2 with random salt and enforces data integrity through SHA-256 hashing. Encrypted files are stored alongside encrypted metadata, ensuring both usability and tamper detection.

Tools Used

- **Python 3.x**
- **PyQt5** – for GUI development
- **Cryptography (Fernet, AES, PBKDF2HMAC)** – for encryption
- **Hashlib** – for SHA-256 hash calculation
- **Argparse** – for CLI options
- **Logging** – for tracking operations
- **OS, JSON, Pathlib** – for file and system handling

Steps Involved in Building the Project

1. **Key Derivation:**
   - Password-based key derivation using PBKDF2 with a random 16-byte salt.
   - Ensures uniqueness and defense against brute-force attacks.
2. **File Encryption:**
   - Reads the original file and computes its SHA-256 hash.
   - Encrypts the content using Fernet (AES-128-CBC with base64 encoding).

   - Stores encrypted data with .enc extension and metadata with .meta.

3. **Metadata Management:**
   - Metadata includes original filename, file size, hash, and timestamp.
   - Stored encrypted to prevent leak of sensitive details.
4. **File Decryption and Integrity Verification:**
   - Decrypts using user-provided password and stored salt.
   - Recalculates hash and compares with stored value to verify integrity.
5. **User Interface (GUI):**
   - Developed using PyQt5 with tabs for "Encrypt" and "Decrypt".
   - Allows file selection, password entry, and progress feedback.
6. **Command-Line Support:**
   - Provides --encrypt and --decrypt options for terminal-based users.

Conclusion

This project successfully demonstrates a secure and user-friendly way to store and manage encrypted files locally. The dual interface (GUI and CLI) makes it versatile for both novice and advanced users. The use of modern cryptographic practices like AES-256, salted key derivation, and file integrity checks ensures the tool meets high standards of security and usability. It can be extended to support cloud uploads, multi-user authentication, or secure sharing features in the future.