Secure File Storage With AES

Abstract

This project demonstrates a secure file encryption and decryption tool built using Python. The system uses AES-256 encryption (via Fernet module from the cryptography library) to encrypt sensitive files. It stores encrypted files with a '.enc' extension and ensures integrity using SHA-256 hash values. This tool offers a simple and practical way to protect confidential files locally.

Introduction

Data security is essential when handling sensitive files, especially on personal or shared systems. This project aims to provide a basic but effective system to securely store files using AES-256 encryption. Python's cryptography module is used to generate keys, encrypt data, and verify file integrity.

Tools Used

- Python 3
- cryptography.fernet (AES Encryption)
- hashlib (for SHA-256 hashing)
- Ubuntu terminal environment
- Optional: CSV for metadata logging

Steps Involved in Building the Project

1. Set up the Python virtual environment and installed required libraries.
2. Wrote a script to generate and save a secure AES key (Fernet).

3. Implemented encryption script to read a file, encrypt it, and store the result as '.enc'.
4. Added SHA-256 hashing to verify file integrity.
5. Created a decryption script to restore the encrypted file.
6. (Optional) Implemented metadata logging with timestamp and hash in a CSV file.

Output & Observation

The screenshots illustrates the successful execution of key project steps:

1.Key Generation:

The AES key was securely generated and saved to aes.key.

2.Encryption Phase:

The user input secret.txt as the file to encrypt.

File was encrypted and saved as secret.txt.enc.

A SHA-256 hash was generated: 6432f513cfd40d47c8584494c0524468257e50dc1a0422f73becac85189543f8

This hash ensures the integrity of the original file.

3.Decryption Phase:

The file secret.txt.enc was decrypted successfully.

Output saved as secret.txt_decrypted.txt.

4.Command Mistake & Recovery:

An incorrect command cat secret_decrypted.txt caused a file-not-found error.

The correct file secret.txt_decrypted.txt was then accessed successfully using the right filename.

This end-to-end flow confirms that the encryption-decryption process works securely and reliably.

Conclusion

This project proved that a lightweight, secure file storage system can be implemented using Python with minimal dependencies. By applying AES encryption and hash-based integrity checking, users can ensure the confidentiality and authenticity of their sensitive files. It’s a great starting point for secure file management.