Secure File Storage System Using AES-256

Abstract

The project implements a local file-encryption system that ensures confidentiality, authenticity, and integrity of user data. It employs the Advanced Encryption Standard (AES-256) in Galois/Counter Mode (GCM) to protect files with strong symmetric encryption. Each file's metadata—name, creation time, and SHA-256 hash—is securely encrypted and stored separately, preventing unauthorized modification. The program was built and executed in Google Colab using Python's *cryptography* library. The outcome is a reliable, portable, and tamper-proof file-protection utility that enables secure storage and retrieval of sensitive documents.

Introduction

With the rise of cloud sharing and data portability, the need for secure local file protection has become crucial. AES-256, a symmetric-key block cipher standardized by NIST, offers an optimal balance between security and performance.
This project demonstrates a simple yet robust implementation of AES-256 encryption and decryption integrated with metadata handling and hash-based integrity checking. The solution prevents unauthorized file access, detects tampering, and ensures that the decrypted output matches the original data exactly.

Tools Used

- **Programming Language:** Python 3
- **Libraries:** cryptography (for AES-GCM encryption/decryption, scrypt KDF)
- **Environment:** Google Colab (uses google.colab.files for upload and download)
- **Algorithms:** AES-256-GCM for authenticated encryption, SHA-256 for hash verification, Scrypt for key derivation

Steps Involved in Building the Project

1. **Environment Setup** – Installed the cryptography package in Colab.

2. **Key Derivation** – Derived a 32-byte AES-256 key from a user password using the scrypt algorithm (salt + cost factors $N = 2^{14}$, $r = 8$, $p = 1$).
3. **Encryption Process**
    - User uploads any local file.
    - A random salt and nonce are generated.
    - The file is encrypted with AES-GCM to produce filename.enc.
    - Metadata (original filename, timestamp, SHA-256 of plaintext) is JSON-encoded and encrypted to filename.enc.meta.enc.
    - A header file (.header.json) stores the salt and KDF parameters needed for decryption.
4. **Decryption Process**
    - User uploads the three components (.enc, .meta.enc, .header.json).
    - The password is used to regenerate the key and decrypt the files.
    - AES-GCM verifies ciphertext integrity; the system recomputes SHA-256 to confirm data authenticity.
5. **Automation and Error Handling**
    - Smart file selector automatically detects the correct .enc file and renames duplicate uploads ((1), spaces).
    - The program outputs a fully restored file (e.g., task1_decrypted.pdf) with "Integrity: PASSED."

Conclusion

The implemented **Secure File Storage System** successfully encrypts, stores, and decrypts files using AES-256 GCM encryption and SHA-256 verification, ensuring both confidentiality and integrity. The combination of password-based key derivation, per-file metadata encryption, and automatic integrity checks provides strong defense against data tampering or unauthorized access.