Title: Secure File Storage System using AES Encryption

Abstract:
This project implements a secure file storage system using AES-based symmetric encryption. It encrypts any file using Python’s cryptography library and generates a secure ciphertext version. Along with encryption, the system stores metadata such as file name, file size, timestamp, path, and a SHA-256 hash to verify file integrity. The decryption module restores the file and confirms whether the data was tampered with. This project demonstrates practical cryptography concepts used in real-world cybersecurity.

Introduction:
Protecting sensitive data is one of the most important challenges in cybersecurity. Files stored locally without encryption can be accessed, modified, or stolen by attackers. AES (Advanced Encryption Standard) is the most widely used encryption standard for secure data protection. In this project, a Python-based file encryption system is implemented using AES-Fernet to ensure confidentiality and integrity. The program encrypts files, stores metadata, verifies tampering using hashes, and decrypts files securely. This project helps understand how encryption and integrity checking work in real-world applications.

Tools Used:

Python 3

cryptography (Fernet – AES)

SHA-256 hashing

JSON (metadata storage)

Kali Linux environment

Objectives:

Encrypt files using AES-based Fernet

Decrypt encrypted files securely

Maintain metadata for encrypted files

Verify integrity using SHA-256 hash

Demonstrate practical secure storage concepts

Steps Involved:

1. The program generates or loads an AES-based Fernet key stored in secret.key.

2. The user selects a file to encrypt.

3. The file data is read and encrypted into ciphertext.

4. The encrypted file is saved with a .enc extension.

5. Metadata such as file name, size, timestamp, and SHA-256 hash is stored in metadata.json.

6. During decryption, the encrypted data is restored to its original form using the same key.

7. A SHA-256 hash is recomputed to verify whether the file has been modified.

8. If the hash matches, the file is confirmed as original; otherwise, it is marked as tampered.

Sample Output:

```
[+] File encrypted: sample.txt.enc
[+] Metadata saved.
Integrity Check: PASSED
Decrypted file: DECRYPTED_sample.txt
```

Conclusion:
The Secure File Storage System using AES encryption successfully demonstrates strong file protection and integrity verification. By combining AES encryption, metadata logging, and SHA-256 hashing, this project provides practical knowledge of cryptography and secure data handling. It meets all the requirements of a cybersecurity internship project and showcases essential skills useful in real-world security applications.