PROJECT REPORT ON
SECURE FILE STORAGE SYSTEM WITH AES

1. Introduction:

In today's digital world, protecting sensitive files is more important than ever. Data breaches, unauthorized access, and file tampering are common cybersecurity threats. To address this, we have developed a Secure File Storage System using AES-256 encryption, which allows users to safely encrypt and store their files locally. The system ensures data confidentiality and integrity using strong cryptographic techniques.

2. Abstract:

This project aims to create a simple yet effective solution for securely storing files on a local machine using AES (Advanced Encryption Standard). The tool provides an intuitive web interface where users can upload files, encrypt them, and later decrypt them using the correct key. Additionally, it includes hash verification to detect any file tampering. The solution is built using Python, Flask, and HTML/CSS for the frontend. The encryption algorithm ensures that even if files are accessed by an attacker, their contents remain unreadable.

3. Tools Used:

- Python – Programming language used to implement AES and backend logic.
- Flask – Lightweight web framework for Python to create the web app.
- Cryptography Library – For AES-256 encryption and decryption.
- HTML/CSS – Used to design the web-based user interface.
- Hashlib – For generating SHA-256 file hashes to verify file integrity.

4. Steps Involved in Building the Project:

1. Setup Project Structure
    - Create project folder with separate directories for uploads, encrypted files, and decrypted files.
    - Install required Python packages: flask, cryptography.
2. Implement AES Encryption
    - Use AES with a 256-bit key and secure mode (CBC or Fernet).
    - Create functions to encrypt and decrypt files using a secret key.
3. Create Flask Backend
    - Set up Flask routes to handle file upload, encryption, and decryption.

- Encrypt uploaded files and save them with .enc extension.
- Decrypt files when the user uploads an encrypted file.

4. Design Web Interface
    - Build a simple HTML/CSS interface with:
        - File input for encryption.
        - File input for decryption.
        - Submit buttons for both actions.
    - Add dark theme with responsive and hover effects.
5. Add Hash Verification
    - After encryption and decryption, compute file hash (SHA-256).
    - Compare original and decrypted file hashes to ensure data wasn't tampered.
6. Test the Application
    - Upload sample files, encrypt and decrypt them.
    - Verify that the decrypted file is the same as the original.

4. Execution:

Run

```
python app.py
```

Visit http://localhost:5000

5. Conclusion:

This project demonstrates a simple yet powerful way to secure files using AES encryption. With an easy-to-use web interface, users can protect their sensitive data from unauthorized access. The addition of file integrity checks ensures that files are not tampered with. This system is ideal for personal use or as a component in larger secure file storage solutions. In conclusion, this tool lets you lock your files safely and unlock them only when needed, keeping them safe from others.