TASK 3 REPORT

Secure File Sharing System (AES Encrypted)

Intern Name:	Shahidul Hasan
CIN:	FIT/JUN2S/CS1850
Domain:	Cyber Security
Internship Provider:	Future Interns
Month:	June 2025
Tools Used:	Python, Flask, PyCryptodome, HTML/CSS

Objective

To build a secure file sharing system where users can upload and download files, with every file encrypted using AES before storage and decrypted securely during download. This project simulates real-world client needs such as secure document transfer in healthcare or legal domains.

Technology Stack

Component	Tech Used
Backend	Python + Flask
Encryption	AES (CBC Mode) via PyCryptodome
Frontend	HTML + CSS
File Storage	Encrypted `.enc` in local folder
Version Control	Git + GitHub
Video Tool	Loom / OBS

System Workflow

1. User uploads a file through a Flask form.
2. Backend reads the file and encrypts it using AES-128-CBC.
3. Encrypted file is saved with a `.enc` extension.
4. All encrypted files are listed on the homepage.
5. User clicks to download → file is decrypted on-the-fly and served.

Encryption Details

Aspect	Info
Algorithm	AES (Advanced Encryption Standard)
Mode	CBC (Cipher Block Chaining)
Key Size	128-bit (16-byte)
IV Handling	Random IV per file, prepended to ciphertext

Padding Scheme	Manual (space-padded to block size)
Key Management	Static key in demo (`securefileaes128`)

Demonstration Summary

• Uploaded files: `task2final.pdf`, `Screenshot.png`
• Files encrypted → stored as: `task2final.pdf.enc`, `Screenshot.png.enc`
• Downloaded file decrypted correctly and matched original
• Logs confirmed proper AES encryption flow

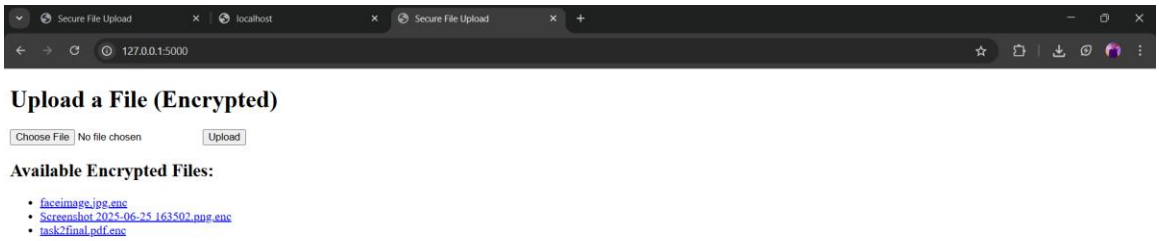
📸 Screenshots

Below are key screenshots demonstrating the system in action:

---

1. File Upload Form

📍 *Location:* Home page (http://127.0.0.1:5000)
📝 *Description:* User interface for uploading any file (PDF, image, document, etc.) that will be encrypted before storage.



2. Encrypted File Listing

📍 *Location:* Home page file list

📝 *Description:* Shows files that have been uploaded and encrypted, saved with .enc extension inside the /uploads folder.

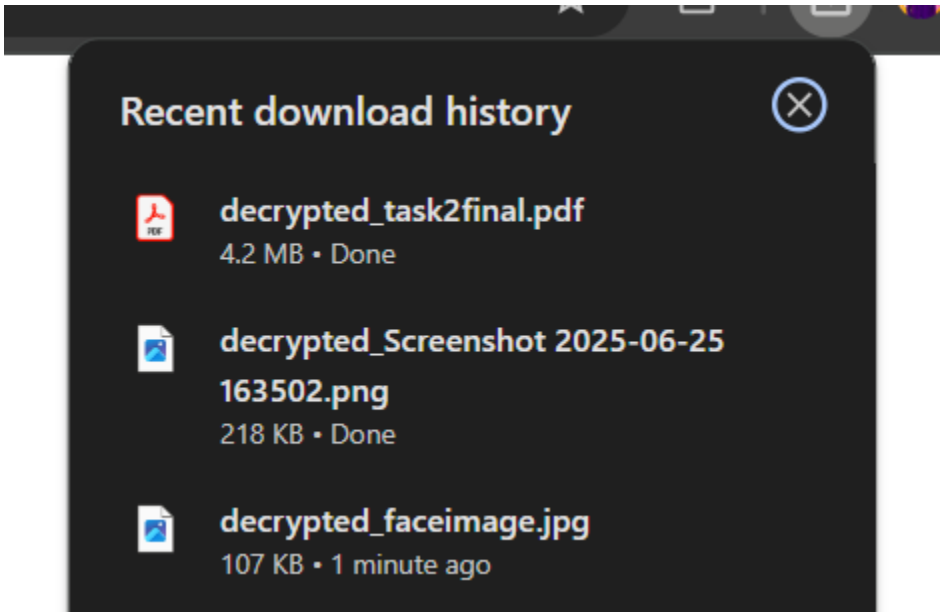
> **Available Encrypted Files:**
>
> - faceimage.jpg.enc
> - Screenshot 2025-06-25 163502.png.enc
> - task2final.pdf.enc

---

3. Decrypted File Download

📍 *Location:* Clicked on file name for download

📝 *Description:* Flask decrypts the .enc file on the fly and downloads the original unencrypted file securely.

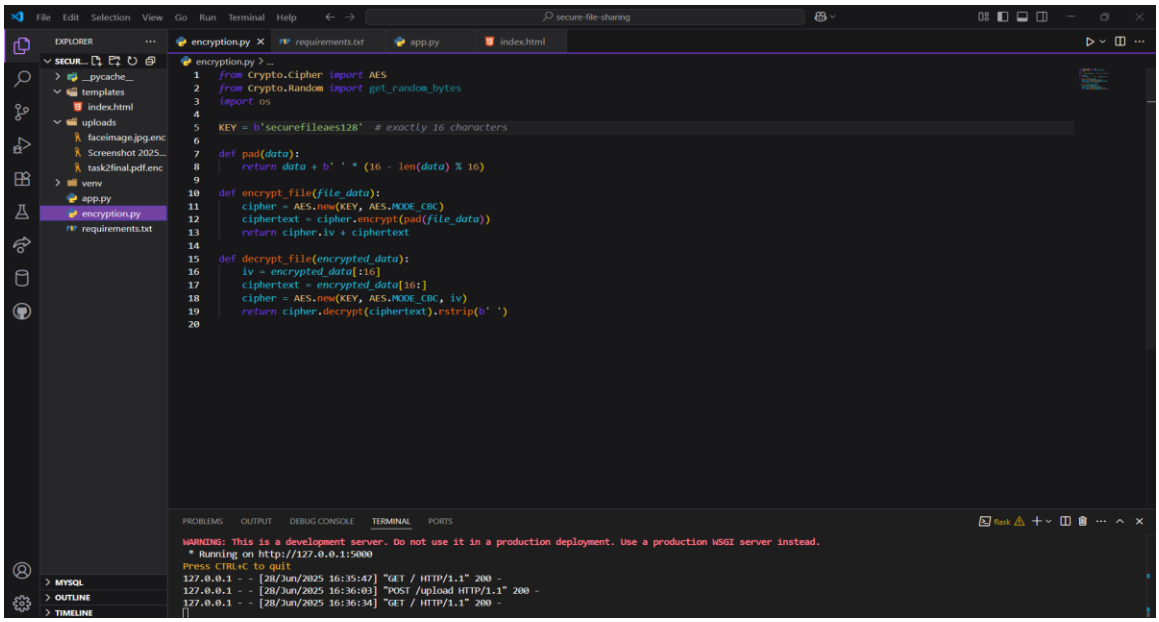


---

4. Encryption Logic in Code

📍 *File:* encryption.py

📝 *Description:* Highlights use of AES-128-CBC mode, padding, IV generation, and secure file handling.

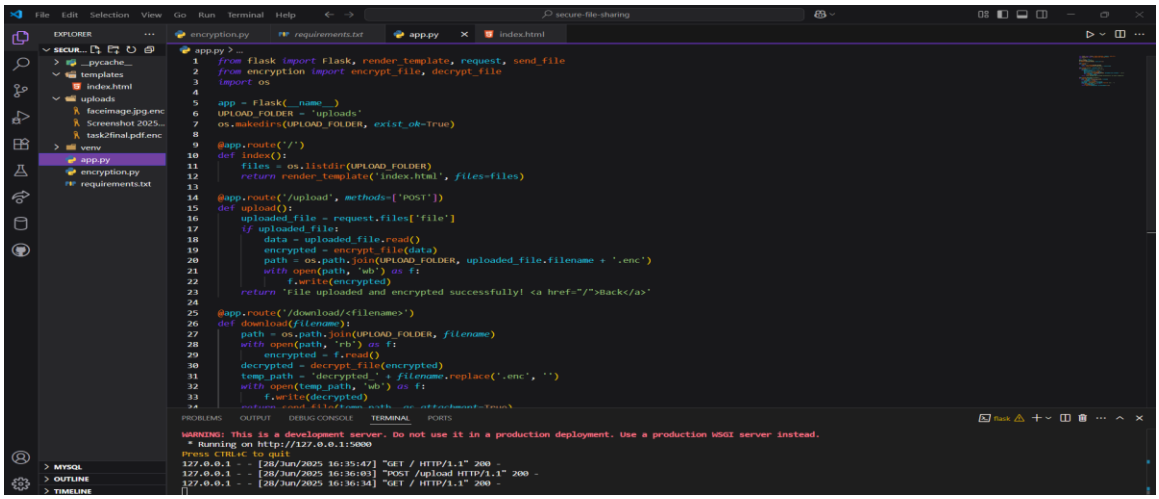


---

5. Flask App Routes (Upload/Download)

📍 *File:* app.py

📝 *Description:* Routes for file upload (/upload) and download (/download/<filename>) that handle encryption and decryption.



Learning Outcomes

- Built an end-to-end secure file handling system
- Implemented AES encryption/decryption manually
- Learned about CBC mode, IV usage, and key handling
- Integrated cryptographic logic into Flask
- Strengthened practical skills in secure web development

🔐 Security Overview

This system implements **AES (Advanced Encryption Standard)** to protect all files uploaded through the Flask web interface.

🔑 Encryption Details

Property	Description
Algorithm	AES (Advanced Encryption Standard)
Mode	CBC (Cipher Block Chaining)
Key Size	128 bits (16 bytes)
Key Used	Static key securefileaes128 (demo purpose)
IV (Nonce)	Random IV generated per file, stored at beginning of ciphertext
Padding	Manual space-padding to align data to 16-byte blocks
Ciphertext	Saved as .enc in /uploads folder

🔓 Decryption Flow

1. Extract IV from first 16 bytes of the encrypted file

2. Use the same AES key and IV to decrypt the content
3. Remove padding
4. Serve decrypted file to the user

🛡️ Security Practices Followed

- No decrypted files stored permanently
- Encryption key stored only in backend (not exposed to frontend or logs)
- IV is generated securely using Crypto.Random.get_random_bytes()
- Decryption only triggered on user request

⚠️ Notes & Future Enhancements

- Key is static for demonstration. In production, this should be:
    - Stored securely in .env or key vault
    - Rotated periodically
- Consider adding:
    - User authentication
    - File expiration or access logs
    - Hash-based file integrity check

Deliverables

- 🎥 Walkthrough Video Link: video

Conclusion

This task demonstrated my ability to implement real-world cryptography using AES and combine it with a secure web application. It simulated actual requirements from enterprise or legal environments where confidentiality of shared files is critical. With this, I've completed Task 3 of the cybersecurity internship with hands-on success.