CYBER SECURITY INTERNSHIP REPORT

FUTURE INTERNS

TASK 3: SECURITY FILE UPLOAD/DOWNLOAD PORTAL WITH AES ENCRYPTION

ASSIGNMENT

BY

ATHANASIUS J.K GADOSEY

Introduction

Task Overview

Task 3: Security File Upload / Download Portal with AES Encryption

Objective of Task

This report outlines the design and development of a secure file upload/download web portal using **Python Flask** and **AES encryption** via the **PyCryptodome** library. The objective is to simulate a real-world secure file transfer system, particularly in sectors where data confidentiality and compliance are crucial, such as finance, healthcare, and law.

The application encrypts files at rest and decrypts them upon download, ensuring data remains protected both in storage and in transit. Key security principles such as symmetric encryption, key handling, and integrity testing were also demonstrated.

Tools & Technologies

- **Backend Framework:** Python Flask
- **Encryption:** AES using PyCryptodome
- **Frontend:** HTML forms (for upload/download interface)
- **Key Handling:** Session-based

Functional Overview

Feature	Description
Upload Portal	Accepts user files and encrypts them using AES before saving to disk
Download Portal	Allows the user to retrieve and decrypt previously encrypted files
AES Encryption	Ensures the confidentiality of life content via symmetric encryption
Key Management	Random key generated on session start
Encrypted Storage	Files are saved with the ".enc" extension in a dedicated directory.

Security Architecture

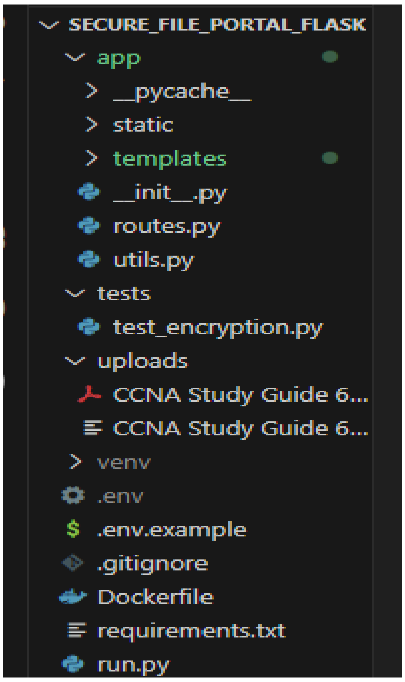
Component	Implementation Details
Encryption	AES-128 CBC mode with random IV; files padded using PKCS7
IV Handling	IV is prepended to the ciphertext during encryption for reuse during decryption
Key Handling	Key generated using **get_random_bytes(16)**; stored in memory for the session only
File Separation	Encrypted and plaintext files are stored in separate directories to prevent mixing
Transport Security	HTTPS can be enabled using Flask + SSL
Input Sanitization	Basic filename validation applied; recommend adding type and size checks in prod

UI Overview

- Upload Form: **POST /upload** with file input field
- Download Form: GET /download?filename=yourfile.enc
- Minimal frontend layout

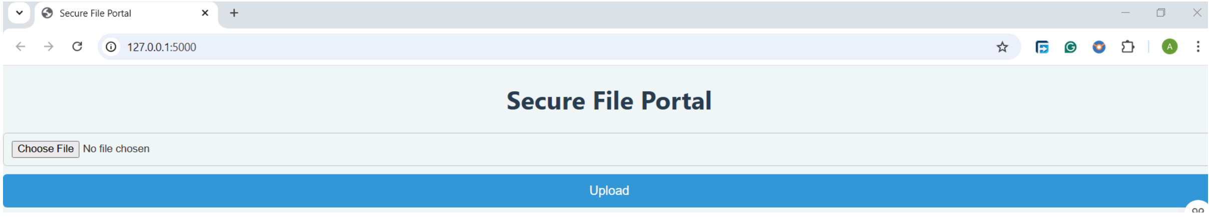
Screenshots

- **Figure 1 – Folder Structure**

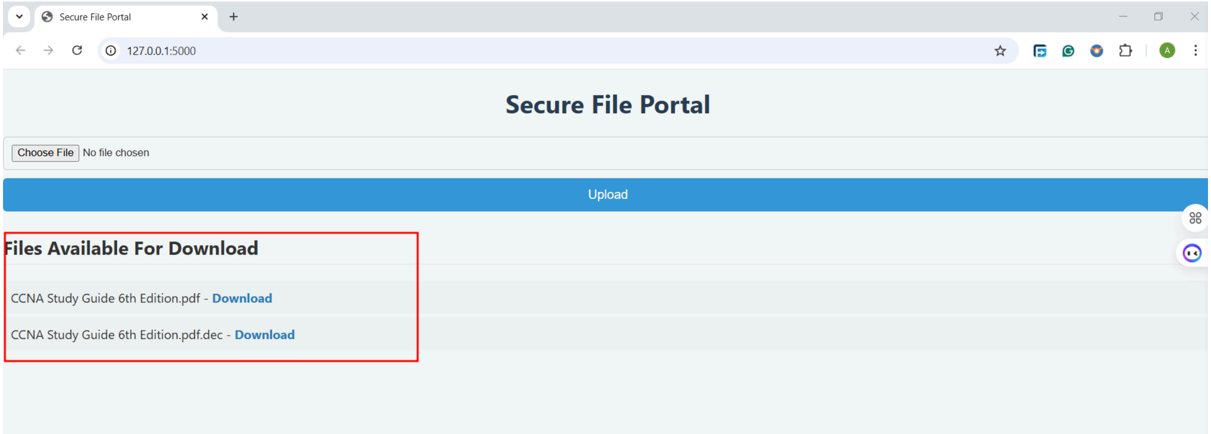


```
SECURE_FILE_PORTAL_FLASK
  app
    __pycache__
    static
    templates
    __init__.py
    routes.py
    utils.py
  tests
    test_encryption.py
  uploads
    CCNA Study Guide 6...
    CCNA Study Guide 6...
  venv
  .env
  .env.example
  .gitignore
  Dockerfile
  requirements.txt
  run.py
```

- **Figure 2 – Upload Form UI**



- **Figure 3 – Encrypted File Listing (CLI or Dashboard View)**



Files Available For Download

CCNA Study Guide 6th Edition.pdf - **Download**

CCNA Study Guide 6th Edition.pdf.dec - **Download**

Testing & Verification

Test Case	Outcome
Upload and encrypt a file	Encrypted file saved successfully
Download and decrypt the file	Matches the original plaintext file
Invalid file or filename	Returns a user-friendly error
Decryption with the wrong key or IV	Decryption fails

Recommendations for Production Use

- Implement **authentication** and access control
- Store encryption keys using a secure service (e.g., AWS KMS
- Enable HTTPS and HSTS headers
- Enforce file upload restrictions (type, size, Virus scanning)
- Implement logging and alerting for file activity

Project Deliverables

- **run.py:** Flask-based web server with AES encryption logic
- **uploads/:** Decrypted file directory
- **encrypted/:** AES-encrypted file directory
- **README.md:** Setup instructions and usage
- **Security_overview.pdf:** Architecture and encryption summary
- **screenshots/**: Folder containing annotated UI and terminal screenshots

Conclusion

This project demonstrates secure file handling fundamentals by applying AES encryption in a realistic Flask web app. It simulates client-facing needs for confidentiality and highlights the developer’s ability to apply secure design principles.