FUTURE_CS_03 — Encrypted File Upload & Download Portal Using AES
Analyst: G Narendran
Program Track: Cyber Security | Future Interns Program

Project Goal

Design and implement a secure web-based platform that enables encrypted file uploads and downloads utilizing AES-256 encryption. The key objective is to ensure data confidentiality both during storage and transmission.

Technology Stack & Tools

- **Backend Framework:** Python Flask
- **Encryption:** AES-256 CBC mode (via PyCryptodome)
- **Frontend:** Bootstrap 5
- **Version Control:** Git & GitHub
- **Deployment:** Render.com (HTTPS-enabled)

Implemented Security Measures

- 256-bit AES encryption using CBC mode
- Unique IV (Initialization Vector) generated per file
- PKCS#7 padding to ensure complete encryption blocks
- Secure key management through environment variables
- Files saved in encrypted form under the `uploads/` directory
- Decryption happens only during download, temporarily stored in `decrypted/`
- Secure deployment with HTTPS on Render

Setup & Execution

Clone the repository and run the app locally:

```
git clone https://github.com/sachinsree47/FUTURE_CS_03.git
cd FUTURE_CS_03
pip install -r requirements.txt
python app.py
```

Open your browser and navigate to ☞ http://localhost:5000

You can upload files → they are encrypted and stored → download triggers decryption.

Key Takeaways

- Applied AES encryption to a real-world web solution

- Managed secret keys securely using environment configurations
- Gained hands-on experience with Flask routes and secure file operations
- Successfully hosted a secure application on Render
- Followed OWASP guidelines to maintain best security practices

---

Limitations & Potential Improvements

Current Gaps:

- Lacks user authentication or access control
- No validation for file type or size
- Files remain indefinitely unless manually removed

Suggested Enhancements:

- Integrate login system with role-based access
- Add automated file expiration/deletion
- Validate file types and enforce size limits
- Encrypt and obfuscate filenames for added security
- Maintain access logs for auditing purposes

Final Thoughts

This project brought together essential aspects of encryption, secure development, and full-stack implementation. It served as a practical demonstration of how strong cryptography and responsible design can be combined to create a secure file handling system.
