🛡️ SECURITY OVERVIEW & PROJECT REPORT

Internship Task: Project 3 – Secure File Sharing System
Intern: Suchetha Baddigam
Domain: Cybersecurity (Future Interns)

📌 1. Objective

To build a secure file sharing system that encrypts files at rest and during download using **AES encryption**, ensuring data confidentiality and protection against unauthorized access.

⚙️ 2. Tools & Technologies Used:

component	Tool/Tech
Backend Framework	Python Flask
Encryption Library	PyCryptome
Algorithm Used	AES(ECB mode,128-bit)
Storage	Local File System
Interface	HTML(Flask template)
Testing Tool	Web Browser

🔐 3. Security Features Implemented

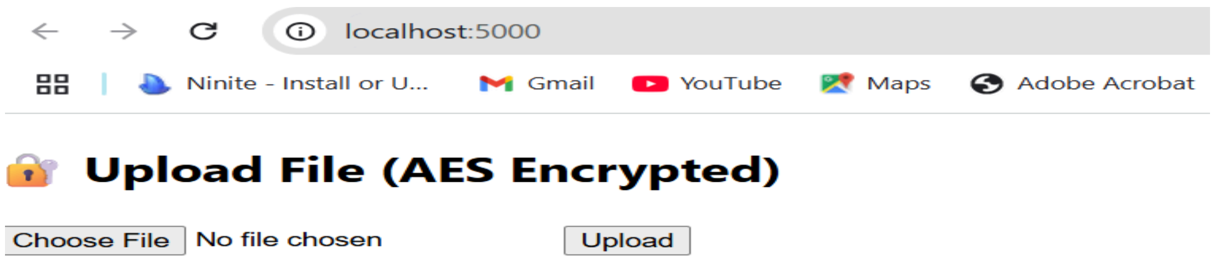
- **AES Encryption:**
  Used AES (Advanced Encryption Standard) in **ECB mode** with a 16-byte secret key to encrypt file content.

- **Upload Protection:**
  File is encrypted immediately after upload before saving to disk.

- **Download Decryption:**
  Files are decrypted on-demand during download to minimize exposure of raw data.

📁 4. How It Works

🔸 **Upload:**



- User selects and uploads a file through the browser.
- Server reads the file → encrypts content using AES → stores it in `uploads/` as `filename.enc`.

🔸 **Download:**

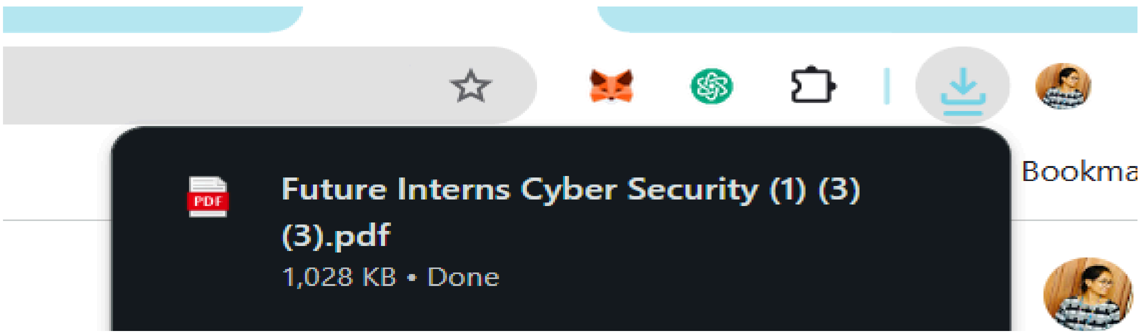
- User visits `/download/<filename>`
- Server reads the `.enc` file → decrypts it using the AES key → sends it to the browser for download.

🔄 5. Sample Input & Output

Upload Example:

```
Uploaded File: Future Interns Cyber Security.pdf

Stored As: uploads/Future Interns Cyber Security.pdf.enc
```



Future Interns Cyber Security (1) (3) (3).pdf
1,028 KB • Done

Download Example:

```
URL: http://localhost:5000/download/Future Interns Cyber
Security.pdf

→ Downloaded File: Decrypted original file
```

🚧 6. Limitations

- Uses **ECB mode**, which is not ideal for real-world secure systems (CBC or GCM is recommended).
- Static key (`b'ThisIsASecretKey'`) is hardcoded; should be stored securely or generated dynamically in production.
- No authentication or access control.

💡 7. Recommendations

- Upgrade to **CBC mode** with random IV for stronger security.
- Implement **user authentication** before allowing uploads/downloads.
- Add file **type and size validation** to prevent abuse.
- Encrypt metadata (e.g., file names) for full confidentiality.

✅ 8. Conclusion

This project successfully demonstrates a secure file handling system using AES encryption to protect uploaded files and safely deliver decrypted versions to authorized users. It reflects foundational concepts in cybersecurity including encryption, confidentiality, and secure web application development.