Security Overview Document

Project Title: Secure File Sharing System
Developer: Suvhankar Dutta
Internship Program: Future Interns – Cyber Security Task 3
Date: July 2025

📘 Overview

The Secure File Sharing System is a Flask-based web application designed to allow users to safely upload and download files over the web. The project focuses on ensuring data confidentiality using **AES encryption**, safe key handling, and secure file transfer practices.

🔑 Encryption & Decryption

🛡️ Algorithm Used: AES (Advanced Encryption Standard)

- **Mode:** ECB (Electronic Codebook Mode)
- **Key Size:** 128-bit (16 bytes)

🔄 How It Works:

- **Encryption on Upload:**
  - Files uploaded via the web interface are read in binary.
  - Padded to match AES block size.
  - Encrypted using a static 16-byte key and saved with .enc extension in the uploads/ directory.
- **Decryption on Request:**
  - Users can input an encrypted file name and download it.
  - Internally, it can be decrypted using the same AES key via decrypt_file() logic.

📁 Relevant Code:

```
KEY = b'mysecretaeskey12'  # 128-bit static key

cipher = AES.new(KEY, AES.MODE_ECB)

encrypted_data = cipher.encrypt(pad(data))
```

🔐 Key Management

- **Current Approach:**
  - A hardcoded static AES key (myscretaeskey12) is stored in encryption.py.
- **Risk:**
  - If the source code is exposed, the key becomes compromised.
- **Suggested Improvement (for production):**
  - Use environment variables to store the AES key securely.
  - Alternatively, use a key management service like AWS KMS, HashiCorp Vault, or Azure Key Vault.

🔍 File Integrity & Safety

- Files are saved **only after successful encryption**.
- Filenames are preserved with .enc appended, ensuring clear distinction.
- Decrypted files are stored in a separate decrypted/ directory to avoid overwriting.

🚨 Vulnerabilities & Limitations

Area	**Risk Description**	**Suggested Fix**
AES ECB Mode	Repetitive patterns can leak structure of data	Use **AES-GCM** or **CBC mode with IV**
Hardcoded Key	Anyone with code access has access to key	Load from environment securely
No Authentication	Anyone accessing the URL can upload/download files	Add **user login** or **token-based access**

Area	Risk Description	Suggested Fix
No HTTPS	Unencrypted transmission over HTTP	Deploy with **SSL/TLS certificate**
No File Size Limit	Users can upload massive files	Add **file size limit** in config
No MIME Type Check	Any file can be uploaded	Validate MIME/file extensions

---

✅ Best Practices Followed

- File uploads handled securely using Flask's request.files
- AES used for secure at-rest file protection
- Encrypted and decrypted files are stored in **separate folders**
- Uses requirements.txt to lock dependencies
- Project structured for clarity and modularity

📈 Future Enhancements

- Switch to **AES-GCM** or **AES-CBC** with IV for better security
- Implement user **authentication and access control**
- Add **file checksum or hash** verification post-decryption
- Deploy over a **secure cloud service** with **HTTPS**
- Log and monitor all file uploads/downloads

📁 References

- PyCryptodome AES Docs
- Flask File Upload Docs
- OWASP File Upload Security Guide
- AES ECB vs CBC

✍️ Author Notes

“This project was built as a hands-on implementation of secure file sharing techniques. It uses industry-grade encryption and showcases practical knowledge of cryptography, Flask backend, and secure coding principles. I plan to upgrade this further by integrating user-level access control and moving to a more secure AES encryption mode.”

Video Link: https://www.linkedin.com/posts/suvhankar-dutta-7890912aa_thrilled-python-aes-activity-7351196108305465344-owMQ?utm_source=social_share_send&utm_medium=member_desktop_web&rcm=ACoAAEpoBBkBbVUtzQ5Y1171e4dMWeZuCKf6pY8

Git Repo: https://github.com/Subhoisalive/FUTURE_CS_03

Author– Suvhankar Dutta