CYBER SECURITY TASK - 3

Security Overview Document – Secure File Sharing System (AES Encryption)

Name: Pinemula Arun Kumar

Organisation: Future Interns

1. Introduction

This security overview explains the encryption model, key management, secure storage, and data flow used in the Secure File Sharing System built with Python Flask and AES encryption (Fernet)**.**

The goal of this document is to clearly **describe** how files are protected at rest and during transfer**,** ensuring confidentiality, integrity, and secure key handling.

---

2. Encryption Method: AES (Fernet)

Your system uses **Fernet**, which internally uses:

- AES-128 in CBC mode
- SHA-256 HMAC for message authentication
- PKCS7 padding
- Base64 encoding for safe storage

Why Fernet?

- Provides **confidentiality** (AES encryption)
- Provides **integrity** (HMAC SHA-256)
- Automatically handles **IV (Initialization Vector)** and **padding**
- Easy and safe to implement
- Avoids common crypto mistakes

3. How Encryption Works in Your System

Step 1 — User Uploads a File

- User selects a file via the web interface
- Flask receives the file via an HTTP POST request

Step 2 — System Encrypts the File

Function used:
from encryption import encrypt_file
Process:

1. Read file bytes
2. Generate AES key (or load existing key)
3. Encrypt using Fernet(key).encrypt(data)
4. Save encrypted file with ".enc" extension

Step 3 — Encrypted File Saved in /uploads

- Clear-text file is never stored
- Only encrypted version is kept on disk

Step 4 — User Downloads an Encrypted File

- Clicking a filename triggers the /download/<file> endpoint
- File still encrypted at this stage

Step 5 — System Decrypts File Before Sending

Function used:
from encryption import decrypt_file
Process:

1. Read encrypted .enc file
2. Use AES key
3. Decrypt using Fernet(key).decrypt()
4. Return clean file to the user

---

4. Key Management

Your system uses a secure key system:

Key Generation

from cryptography.fernet import Fernet

key = Fernet.generate_key()

Key Storage

- The key is stored in secret.key
- File permission is restricted
  (in production: chmod 600 secret.key)

How the key is used

```
with open("secret.key", "rb") as f:
    key = f.read()
```

Important Security Notes

- The key must **never** be uploaded to GitHub
- Key must be rotated periodically
- In a real production system:
  - Use AWS KMS / Azure Key Vault
  - Store key in environment variables

---

5. Secure Architecture Overview

Data Flow Diagram

User → Upload File → Flask Server → AES Encrypt → Save encrypted file (.enc)
User → Request Download → Flask Server → AES Decrypt → Deliver original file

Components:

Component	Security Role
Flask Backend	Handles secure file transfer
AES Encryption (Fernet)	Ensures confidentiality & integrity
secret.key	Master key for encryption
uploads/	Stores encrypted files only
HTML interface	Simple and secure UI for users

---

6. Security Controls Implemented

Security Feature	Description
AES Encryption (Fernet)	Protects files at rest and transit
HMAC Integrity Check	Ensures file not modified
Key Protection	Key stored in secure file

Security Feature	Description
File Type Handling	Prevents execution of dangerous files
Server-side Decryption	User receives only decrypted output
No Plaintext Storage	System stores encrypted files only

---

7. Threat Model

Threat	Mitigation
Unauthorized file access	AES encryption, secure key
Key exposure	Key stored separately & not in GitHub
Tampered encrypted files	Fernet HMAC verification
File injection attacks	Sanitized filename + server-side validation
Man-in-the-middle	Can be prevented via HTTPS (GitHub codespaces already HTTPS)

---

8. Recommendations for Production Deployment

Not required for internship, but useful:

- Use **environment variables** for key
- Enable **HTTPS**
- Add **user authentication**
- Scan files with antivirus before encryption
- Log all file upload/download activities
- Implement **rate limiting**
- Store encryption keys in KMS (AWS, Azure, GCP)

---

9. Conclusion

Your Secure File Sharing System successfully implements:

1. AES Encryption (Fernet)
2. Secure key generation & storage
3. Fully encrypted file storage
4. Server-side decryption
5. Clean and simple UI
6. Strong integrity protection