Internship Project Report

Project Title: Secure File Sharing System Using Python Flask

Intern Name: Rohith Reddy Gundreddy

Internship Duration: 30/08/2025 - 30/09/2025

Company: Future Interns

1. Project Overview

The project implements a secure file portal where users can upload and download files safely. All files are encrypted using AES-based encryption before storage to ensure confidentiality and integrity.

Objectives:

- Build a file upload/download system with encryption.

- Implement secure key management.

- Provide a simple and intuitive web interface.

- Ensure file integrity and security through testing.

2. System Architecture:

```
file_portal/
│── app.py           # Main Flask application
│── secret.key       # AES key stored securely (auto-generated)
│── uploads/         # Encrypted files storage
└── templates/       # HTML templates
    ├── index.html     # Upload form + file list
    └── files.html     # Optional separate file list
```

Workflow:

1. Upload → User selects a file → Flask encrypts → Saves .enc file in uploads/.

2. Download → User requests file → Flask decrypts → Sends original file.

3. Key Management → AES key stored securely in secret.key.

3. Tools and Technologies

- Programming Language: Python 3.x
- Web Framework: Flask
- Encryption Library: cryptography (AES / Fernet)
- Frontend: HTML, Bootstrap
- Environment: .env file (optional for storing key)
- Testing: Python scripts using requests and hashlib

3. Code Overview

app.py contains the main Flask application, handles routes for upload, download, and file listing. It encrypts files before storage and decrypts on download. AES key is stored in secret.key.

HTML templates (index.html) provide upload form and file list.

4. Security Measures

1. Encryption: AES-GCM ensures files are encrypted before storage.

2. Key Management: Key stored in secret.key; automatically generated if missing.

3. File Integrity: Decryption fails if the file is modified or key is incorrect.

4. Safe Filenames: Filenames sanitized with secure_filename.

5. Controlled Access: Files served only via Flask routes.

5. Testing

- Upload multiple file types (text, PDF, image).

- Download files and compare SHA-256 checksum → Must match.

- Inspect uploads/ → Files appear as random binary data.

- Delete key → Download fails → Security verified.

7. Conclusion

Project successfully implements a secure file portal:

- Upload and download functionality

- AES-based encryption

- Secure key management

- File integrity validation

- Simple web interface

Skills Learned: Flask development, AES encryption, secure key handling, testing secure systems.