



# Secure File Sharing System using AES Encryption

**Report Title:** Secure File Sharing System

**Internship Batch:** Future Interns

**Report By:** Ashwanth P.G

## Executive Summary

This project demonstrates a secure file-sharing application built with Python and Flask, designed to ensure the confidentiality and integrity of files shared over the web. Using AES symmetric encryption (via Fernet from the `cryptography` library), the system encrypts files during upload and decrypts them upon download. The user interface is simple and intuitive, enhancing usability without compromising security.

## Objective

- ☐ Prevent unauthorized access to uploaded files.
- ☐ Implement secure file transmission using encryption.
- ☐ Provide a basic, extensible system suitable for internal or small-scale secure file exchange.

## Tools & Technologies Used

- Python: Core programming language
- Flask: Lightweight web framework
- cryptography: Secure AES encryption/decryption
- HTML/CSS: Front-end interface

## Security Features Implemented

- AES Encryption (Fernet): Files are encrypted with a secure symmetric key using AES before storage.
- Key Management: Encryption key is generated and stored in `key.key`, and should be securely managed.
- Secure Upload/Download: Only users with knowledge of filenames can download/decrypt the files.
- Input Validation: File extensions and inputs are sanitized in a real-world deployment.

## Functionality & Workflow

### Upload Workflow:

1. User selects a file via the upload form.
2. File is encrypted using AES and saved in the `uploads/` folder.
3. User receives a confirmation and encrypted filename.

### Download Workflow:

1. User inputs the encrypted filename (e.g., `doc.txt.enc`).
2. The server decrypts the file and returns it as a download.

## Directory Structure

secure\_file\_sharing/

├── app.py

├── generate\_key.py

├── key.key

├── templates/

| ├── index.html

| └── download.html

├── static/

| └── style.css

├── uploads/

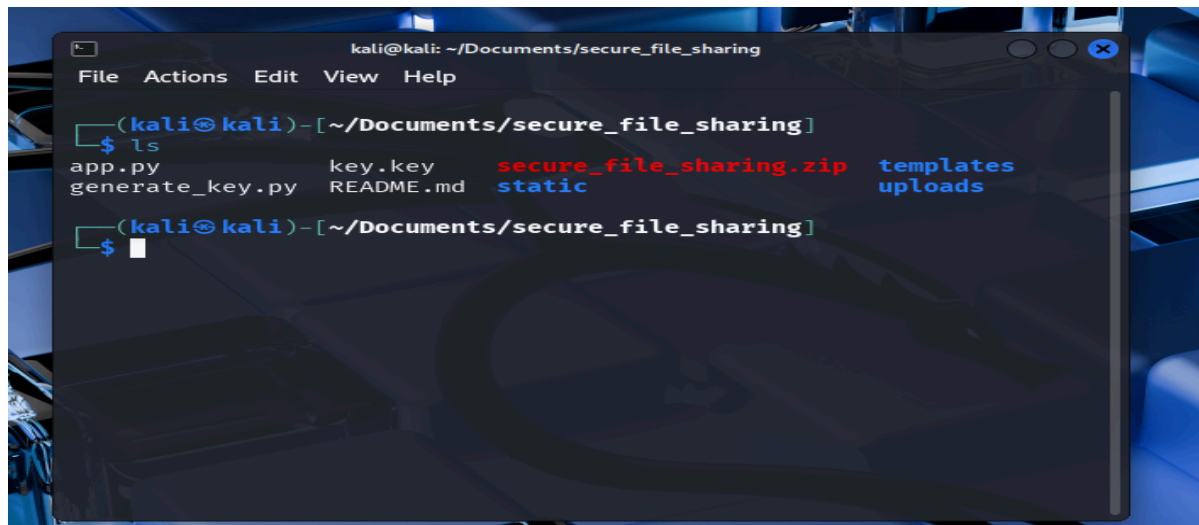
└── [README.md](#)

## Potential Threats & Mitigation

- ★ Key Leakage: Store `key.key` in a secure vault or environment variable.
- ★ File Injection (Path Traversal): Sanitize and validate file names.
- ★ Unauthorized Downloads: Implement user authentication and token-based access.
- ★ No HTTPS Encryption: Use HTTPS in production to avoid MITM attacks.

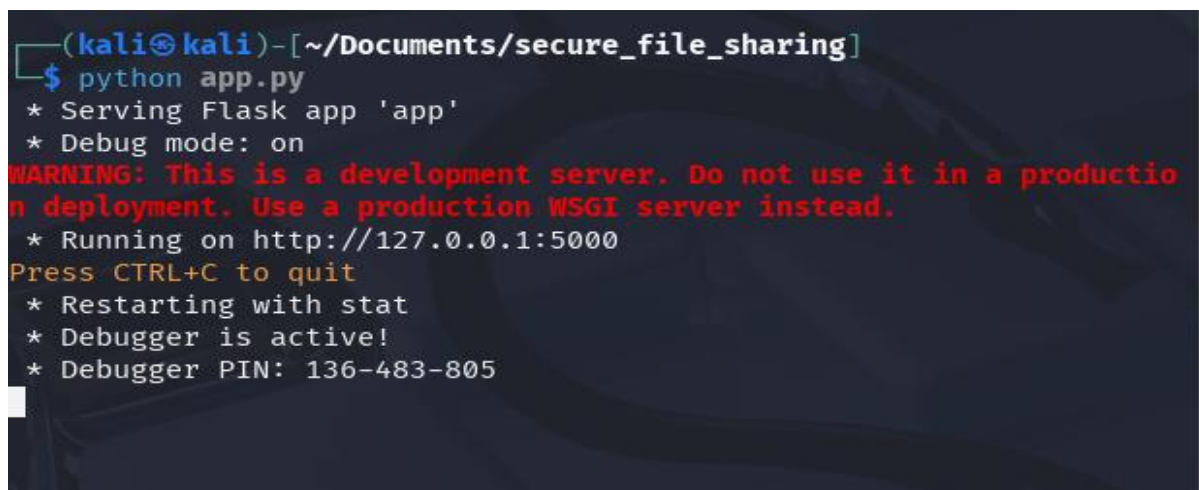
## Sample Output Screens

**1.Directory folder:** contains python backend file and website file with encryption method

A terminal window titled 'kali@kali: ~/Documents/secure\_file\_sharing' with a menu bar (File, Actions, Edit, View, Help). The prompt is '(kali@kali)-[~/Documents/secure\_file\_sharing]'. The user enters '\$ ls' and the output is: 'app.py', 'generate\_key.py', 'key.key', 'README.md', 'secure\_file\_sharing.zip', 'static', 'templates', and 'uploads'.

```
(kali@kali)-[~/Documents/secure_file_sharing]
$ ls
app.py          key.key        secure_file_sharing.zip  templates
generate_key.py README.md      static                  uploads
```

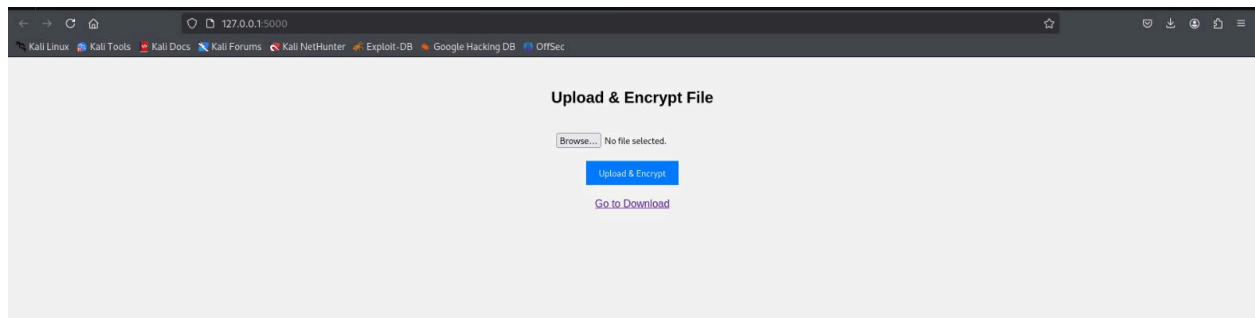
Run the [app.py](#) python file to execute the program and the website hosted in the localhost <http://127.0.0.1:5000> open with web browser

A terminal window titled 'kali@kali: ~/Documents/secure\_file\_sharing' showing the execution of 'python app.py'. The output includes: '\* Serving Flask app 'app'', '\* Debug mode: on', a red warning message 'WARNING: This is a development server. Do not use it in a production n deployment. Use a production WSGI server instead.', '\* Running on http://127.0.0.1:5000', 'Press CTRL+C to quit', '\* Restarting with stat', '\* Debugger is active!', and '\* Debugger PIN: 136-483-805'.

```
(kali@kali)-[~/Documents/secure_file_sharing]
$ python app.py
* Serving Flask app 'app'
* Debug mode: on
WARNING: This is a development server. Do not use it in a production
n deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
* Restarting with stat
* Debugger is active!
* Debugger PIN: 136-483-805
```

## Upload Page: File selection and submission

As we can see here the website for upload and encrypt file also download and decrypt the file

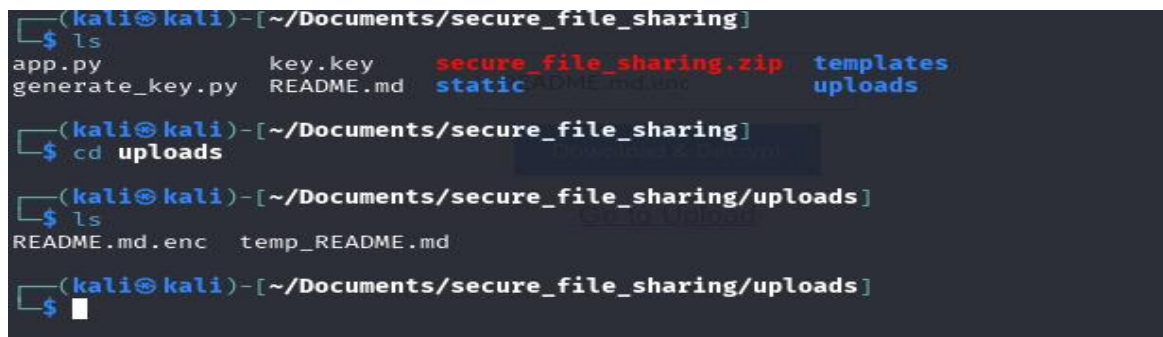


Here we have uploaded the README.md file for sample and it shows like file encrypted and saved as README.md.enc



## Download Page: Input encrypted filename and download

Here we can download and decrypt the file uploaded in the server by typing the encrypted filename and click download. The upload and download log files will be stored in the directory called **uploads**



## **Future Enhancements**

- Add user authentication (JWT or OAuth)
- Integrate file sharing with email or token-based access
- Allow expiration-based file access
- Connect to a database for file tracking

## **Conclusion**

The Secure File Sharing project demonstrates a practical implementation of cybersecurity principles using Python to ensure the confidentiality, integrity, and authenticity of file transfers. By incorporating cryptographic techniques such as AES encryption for securing file content, RSA for secure key exchange, and hashing for integrity verification, the project effectively mitigates common security threats like data interception, tampering, and unauthorized access.

This solution provides a lightweight, cross-platform, and easily extensible framework for secure file transmission over insecure networks. It also highlights the importance of secure key management and user authentication in protecting sensitive information.

Overall, this project not only strengthens the understanding of encryption and secure communication in Python but also lays a foundation for developing real-world secure data exchange systems in both personal and enterprise environments.