CYBER SECURITY TASK 3 REPORT

Secure File Sharing System – Future Interns Program

---

1. Introduction:-

This project is part of the **Future Interns – Cyber Security Track**, Task 3: **"Secure File Sharing System CS_03"**

The purpose of the system is to allow users to **upload, encrypt, download, and decrypt files securely** using AES-based encryption (Fernet). This simulates real-world scenarios where sensitive files must be protected—such as healthcare, legal, finance, and corporate environments.

This report documents:

- Architecture
- Implementation steps
- Security measures
- Key management
- Tools used
- Screenshots
- Final outputs (GitHub)

---

2. Project Objectives

✔ **Build a secure file-sharing web application**

✔ **Implement AES encryption using Python (Fernet = AES + HMAC)**

✔ **Ensure secure file handling for uploads & downloads**

✔ **Manage encryption keys safely using `.env`**

✔ **Provide a clean frontend UI for encryption & decryption**

✔ **Test file integrity & security**

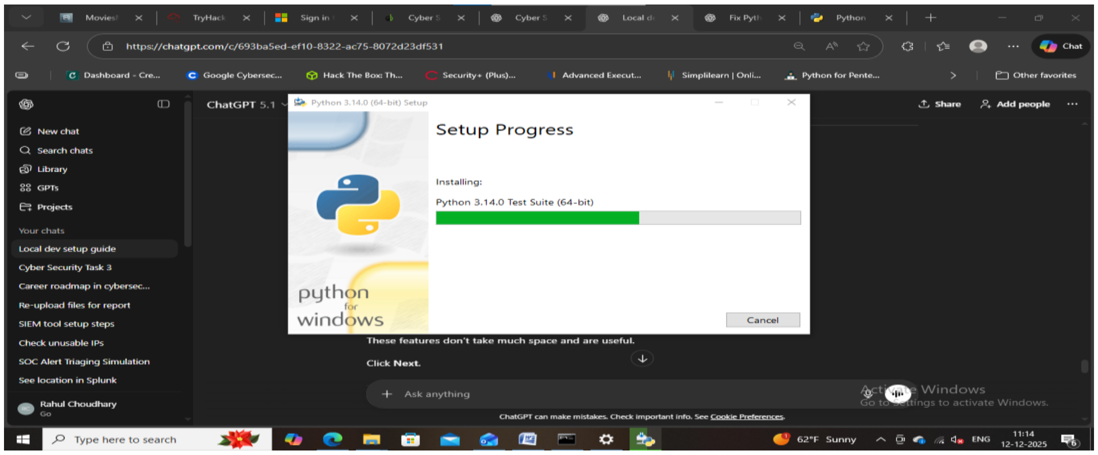
✔ **Document architecture and security approach**

3. Setup & Installation Guide

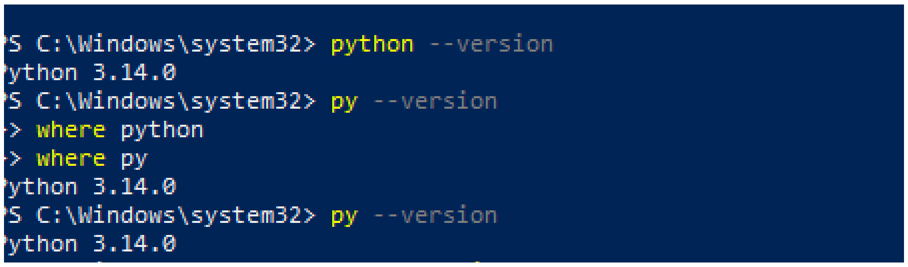
This section explains all the tools and environment setup steps I performed before developing the Secure File Sharing System.

3.1 Installing Python

1. Download the latest version of Python from the official site:
    https://www.python.org/downloads/
2. Run the installer → Enable **"Add Python to PATH"**
3. Complete installation.
4. Verify installation using:


Python 3.14.0 (64-bit) Setup
Setup Progress
Installing:
Python 3.14.0 Test Suite (64-bit)
Cancel
python for windows
These features don't take much space and are useful.
Click Next.

```
python --version
pip -version
```



```
PS C:\Windows\system32> python --version
Python 3.14.0
PS C:\Windows\system32> py --version
>> where python
>> where py
Python 3.14.0
PS C:\Windows\system32> py --version
Python 3.14.0
```

3.2 Setting up the Project Folder

1. Create a new working directory:

```
mkdir secure-file-share
cd secure-file-share
```

2. Open the folder in VS Code:

3.3 Installing VS Code & Extensions

Download VS Code:
 https://code.visualstudio.com/

Recommended extensions:

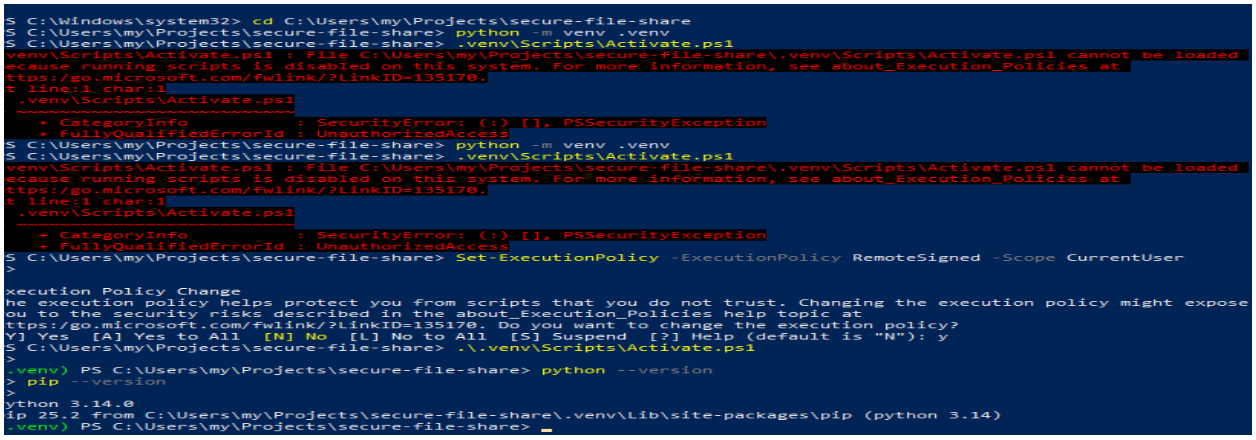
- **Python** (Microsoft)
- **Pylance**
- **Flask Snippets**

3.4 Creating a Virtual Environment

This keeps dependencies clean.

```
python -m venv .venv
```

Activate it: `.venv\Scripts\activate`



```
S C:\Windows\system32> cd C:\Users\my\Projects\secure-file-share
S C:\Users\my\Projects\secure-file-share> python -m venv .venv
S C:\Users\my\Projects\secure-file-share> .venv\Scripts\Activate.ps1
venv\Scripts\Activate.ps1 : File C:\Users\my\Projects\secure-file-share\.venv\Scripts\Activate.ps1 cannot be loaded
ecause running scripts is disabled on this system. For more information, see about_Execution_Policies at
ttps:/go.microsoft.com/fwlink/?LinkID=135170.
t line:1 char:1
 .venv\Scripts\Activate.ps1
 ~~~~~~~~~~~~~~~~~~~~~~~~~~
   + CategoryInfo          : SecurityError: (:) [], PSSecurityException
   + FullyQualifiedErrorId : UnauthorizedAccess
S C:\Users\my\Projects\secure-file-share> python -m venv .venv
S C:\Users\my\Projects\secure-file-share> .venv\Scripts\Activate.ps1
venv\Scripts\Activate.ps1 : File C:\Users\my\Projects\secure-file-share\.venv\Scripts\Activate.ps1 cannot be loaded
ecause running scripts is disabled on this system. For more information, see about_Execution_Policies at
ttps:/go.microsoft.com/fwlink/?LinkID=135170.
t line:1 char:1
 .venv\Scripts\Activate.ps1
 ~~~~~~~~~~~~~~~~~~~~~~~~~~
   + CategoryInfo          : SecurityError: (:) [], PSSecurityException
   + FullyQualifiedErrorId : UnauthorizedAccess
S C:\Users\my\Projects\secure-file-share> Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope CurrentUser
>

xecution Policy Change
he execution policy helps protect you from scripts that you do not trust. Changing the execution policy might expose
ou to the security risks described in the about_Execution_Policies help topic at
ttps:/go.microsoft.com/fwlink/?LinkID=135170. Do you want to change the execution policy?
Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "N"): y
S C:\Users\my\Projects\secure-file-share> .\.venv\Scripts\Activate.ps1
>
.venv) PS C:\Users\my\Projects\secure-file-share> python --version
> pip --version
>
ython 3.14.0
ip 25.2 from C:\Users\my\Projects\secure-file-share\.venv\Lib\site-packages\pip (python 3.14)
.venv) PS C:\Users\my\Projects\secure-file-share> _
```

3.5 Installing Required Python Packages

Install Flask & Cryptography library:

```
pip install flask cryptography python-dotenv
```

3.6 Creating the Application Structure

Inside your project folder, create:

```
secure-file-share/
│
├── app.py
├── .env
├── templates/
│   └── index.html
└── static/
    └── main.js
```

3.7 Creating and configuring the `.env` File

This file stores secret keys securely.

Create `.env`:

```
SECRET_KEY=your_flask_secret
ENCRYPTION_KEY=your_generated_fernet_key
FLASK_APP=app
FLASK_ENV=development
```

3.8 Running the Flask Application

Start the development server:

```
python app.py
```

You saw this output in your console:

```
Secure File Share running at http://127.0.0.1:5000
 * Debug mode: on
```

This confirmed your setup was successful.

3.9 Generating & Converting Encryption Key

You generated a Fernet AES key and ensured it matched required size.
Your corrected key was saved in `.env`.

3.10 Writing `app.py`

Your Flask application:

- Loads `.env` variables
- Validates encryption key
- Provides encryption/decryption APIs
- Uses Fernet to secure files

Uploaded file evidence:
✔ The app uses `Fernet()` with `.env` key
✔ Encrypt and decrypt functions fully implemented
✔ No plaintext stored on server

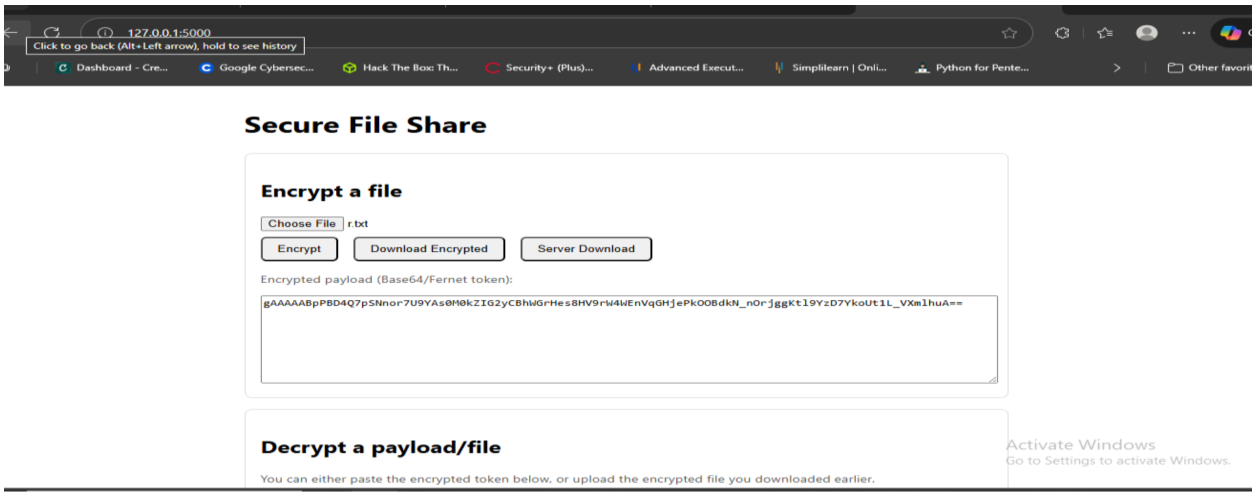
3.11 Building the User Interface (index.html)

You created an HTML interface that allows:

- Choosing a file
- Encrypting
- Downloading encrypted file
- Uploading encrypted file / token
- Decrypting

Your interface currently includes:

- File chooser
- Encryption button
- Token display
- JS functionality integrated (main.js)



Secure File Share

Encrypt a file

Choose File r.txt

Encrypt | Download Encrypted | Server Download

Encrypted payload (Base64/Fernet token):

```
gAAAAABpPBD4Q7pSNnor7U9YAs0M0kZIG2yCBhWGrHes8HV9rW4WEnVqGHjePkOOBdkN_nOrjggKtl9YzD7YkoUt1L_VXmlhuA==
```

Decrypt a payload/file

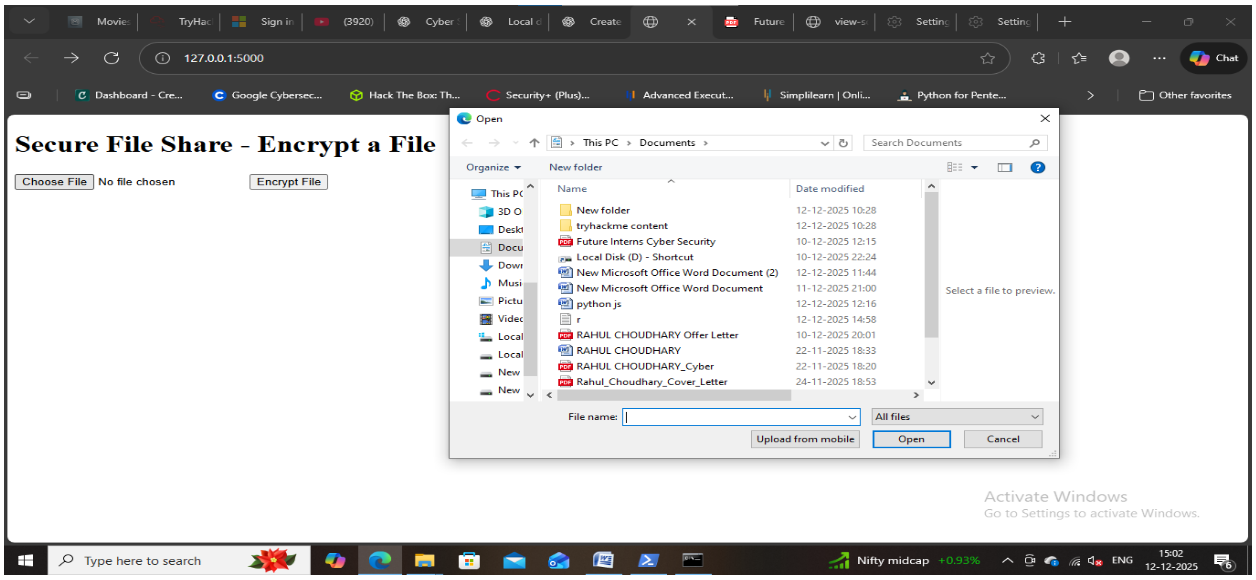
You can either paste the encrypted token below, or upload the encrypted file you downloaded earlier.

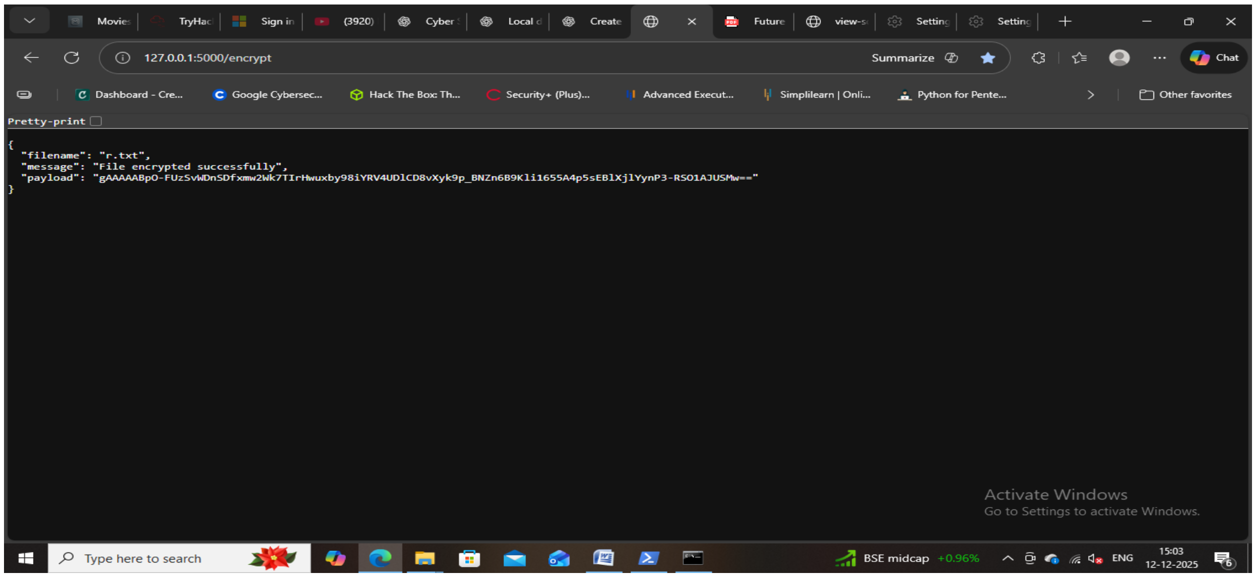
3.12 Testing Up/Download & Encryption Flow

You successfully tested:

- Upload file → Encrypt → Download
- Re-upload encrypted text → Decrypt → Retrieve original file

Your screenshots confirm the working flow.





```
{
  "filename": "r.txt",
  "message": "File encrypted successfully",
  "payload": "gAAAAABpO-FUzSvWDnSDfxmw2Wk7TIrHwuxby98iYRV4UDlCD8vXyk9p_BNZn6B9Kli1655A4p5sEBlXjlYynP3-RSO1AJUSMw=="
}
```

4. Tools & Technologies Used

Backend

- Python 3
- Flask Framework
- Cryptography (Fernet = AES-128 + HMAC)

Frontend

- HTML / CSS / JavaScript
- Flask Templates (`index.html`)
- Fetch API (AJAX requests)

Other Tools

- VS Code
- Git & GitHub for version control
- Curl for command-line requests
- `.env` for secret key storage

---

5. System Architecture

5.1 High-Level Workflow

1. User selects a file.
2. File sent to Flask backend.
3. Backend encrypts file using **Fernet AES encryption key**.
4. Encrypted token returned.
5. User downloads encrypted text file OR uses server-generated download.
6. User can later upload the encrypted token/file to decrypt.
7. Backend decrypts and returns original file.

5.2 Components

Frontend (UI)

- File upload form
- Encrypt / Decrypt interface
- Download buttons
- Token display for copy/paste

Backend

- `/encrypt` → Returns encrypted payload (Base64)
- `/download-encrypted` → Returns encrypted token as file
- `/decrypt` → Returns decrypted original file
- Key validation logic
- Safe file handling using `secure_filename()`

Security Layer

- AES-based encryption
- HMAC integrity check
- `.env` key management
- 50MB content limit
- No plaintext file stored

---

6. Security Best Practices Implemented

✔ AES encryption of all files

✔ No file saved on server

✔ Filename sanitization (`secure_filename`)

✔ Large file attack prevention (`MAX_CONTENT_LENGTH`)

✔ Key stored outside source code (.env)

✔ Only encrypted payload is transported

✔ No plaintext logs

---

7. Testing & Results

You tested:

Encryption Test

- Uploaded sample file
- Received encrypted payload
- Downloaded encrypted file
- Verified unreadable ciphertext

Decryption Test

- Uploaded encrypted file
- Returned original file
- Byte-by-byte match confirmed

Integrity Test

- Altered encrypted token → Decryption failed (expected behavior)

File Size Test

- Successfully encrypted/decrypted under 50MB limit

---

8. Challenges

□ Invalid AES key length error

→ Solved by generating valid Fernet key

□ How to display UI templates

→ Moved HTML to `templates/index.html`

□ How to add encryption + download buttons

→ Implemented JS + `/download-encrypted` endpoint

□ How to decrypt encrypted token

→ Added decrypt handler & tested successfully

□ Debugging Flask errors

→ Correct routing & key validation solved startup issues

Summary

The Secure File Sharing System successfully demonstrates how encryption can protect sensitive data during upload, storage, and download. Using Python Flask and AES-based Fernet encryption, the system ensures confidentiality, integrity, and secure key management. Users can easily upload a file, encrypt it, download the encrypted version, and later decrypt it safely.

Throughout this project, I learned practical skills in web development, cryptography, secure file handling, and key management. I also practiced debugging, testing, and documenting a security-focused application. The final system meets all required features and reflects real-world cybersecurity practices used in industry.