SECURE FILE SHARING SYSTEM REPORT

(Task-3 – Cyber Security Internship Project)

---

1. Introduction

This project focuses on building a **Secure File Sharing Web Application** that allows users to upload and download files safely. The primary objective is to ensure that all uploaded files are **encrypted before storage** and **decrypted only during authorized downloads**, thereby protecting sensitive data from unauthorized access.

This system simulates real-world secure data sharing requirements commonly used in healthcare, legal, and corporate environments.

---

2. Objectives

- To develop a secure web-based file sharing platform
- To implement **AES-256 encryption** for files at rest
- To ensure secure file upload and download operations
- To demonstrate basic cryptographic key management
- To follow secure coding and data protection best practices

---

3. Technology Stack

Component	**Technology**
Backend	Python Flask
Encryption	AES-256 (PyCryptodome)
Frontend	HTML with Embedded CSS
Key Management	Secure Random Key Generation
Tools	VS Code, Git, GitHub

---

4. System Architecture

The system follows a simple and secure architecture:

1. User uploads a file via the web interface
2. File is encrypted using AES-256 before storage
3. Encrypted file is stored in the server's upload directory
4. User requests file download
5. File is decrypted on the server
6. Original file is securely downloaded by the user

---

5. Encryption & Security Implementation

Encryption Method

- **AES-256 (Advanced Encryption Standard)** using CBC mode
- A unique Initialization Vector (IV) is generated for each file

Key Management

- Encryption key is generated using a cryptographically secure random generator
- Key is stored securely in a separate secret.key file
- Key is never exposed to the frontend or hardcoded in the application

File Protection

- No plaintext files are stored on the server
- Only encrypted .enc files exist in the upload directory

---

6. Features Implemented

- Secure file upload functionality
- Automatic file encryption before storage
- Secure file download with on-the-fly decryption
- Clean and user-friendly web interface
- Proper error-free encryption and decryption flow

---

7. Testing & Validation

The application was tested using multiple file types to ensure:

- Files are correctly encrypted after upload
- Encrypted files cannot be read directly
- Downloaded files match the original content
- System handles multiple uploads correctly

All tests were successfully completed.

---

8. Security Best Practices Followed

- Use of strong encryption (AES-256)
- Secure cryptographic library (PyCryptodome)
- Separation of cryptographic logic
- No hardcoded secrets
- Controlled access to files

---

9. Screenshots & Demonstration

- File upload interface
- Encrypted files stored on server
- Successful file decryption and download
- Application execution recording

(Screenshots and screen recording attached separately)

---

10. Conclusion

The Secure File Sharing System successfully demonstrates how encryption can be used to protect sensitive data during storage and transfer. By implementing AES-256 encryption and secure key management, the system ensures confidentiality and data integrity. This project provides hands-on experience with cryptography, secure file handling, and real-world web application security concepts.

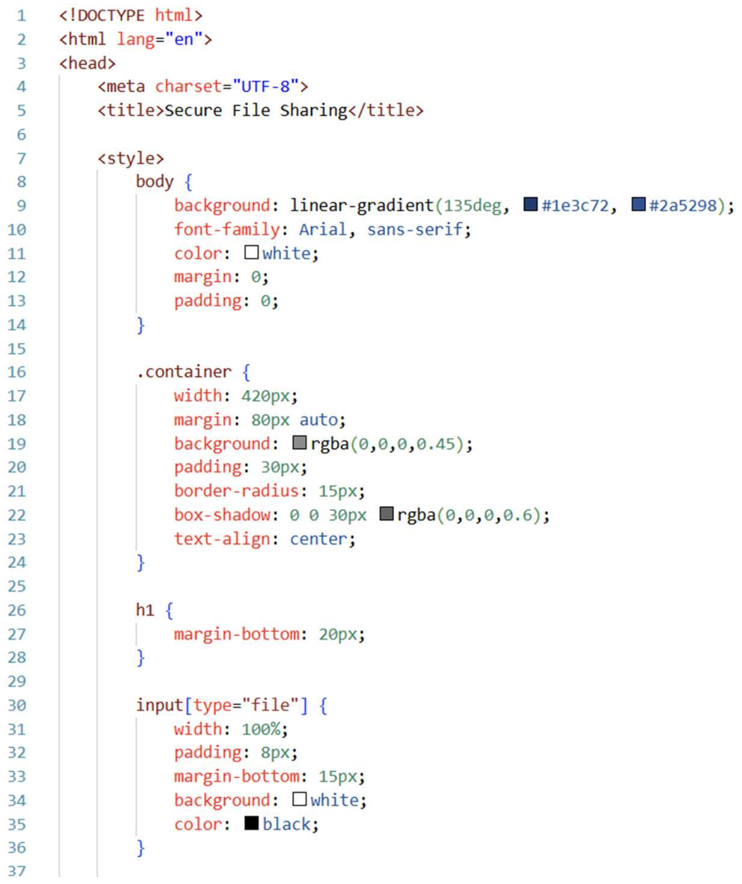
---

11. Future Enhancements

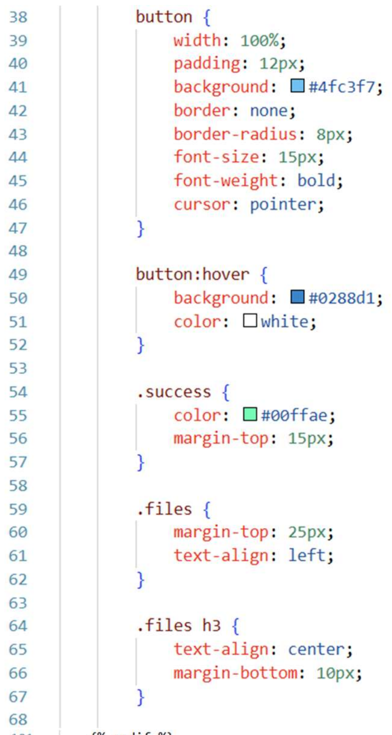
- User authentication and authorization
- Role-based access control
- Secure key vault integration
- HTTPS implementation
- Audit logging and monitoring

12. Evidences

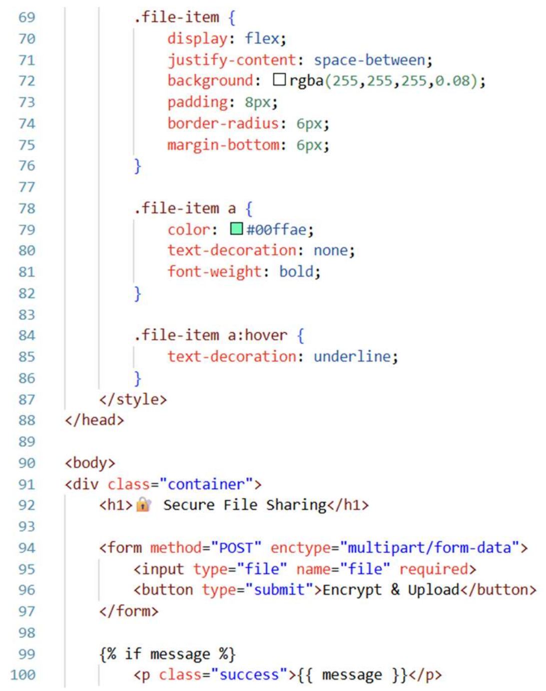
1. HTML frontend



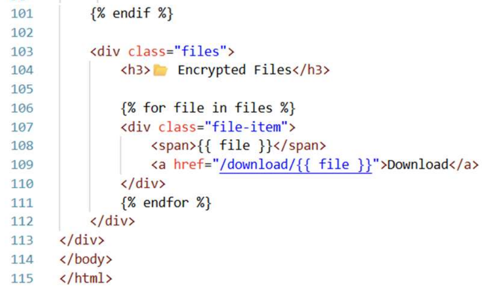
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Secure File Sharing</title>

    <style>
        body {
            background: linear-gradient(135deg, #1e3c72, #2a5298);
            font-family: Arial, sans-serif;
            color: white;
            margin: 0;
            padding: 0;
        }

        .container {
            width: 420px;
            margin: 80px auto;
            background: rgba(0,0,0,0.45);
            padding: 30px;
            border-radius: 15px;
            box-shadow: 0 0 30px rgba(0,0,0,0.6);
            text-align: center;
        }

        h1 {
            margin-bottom: 20px;
        }

        input[type="file"] {
            width: 100%;
            padding: 8px;
            margin-bottom: 15px;
            background: white;
            color: black;
        }

```



```
        button {
            width: 100%;
            padding: 12px;
            background: #4fc3f7;
            border: none;
            border-radius: 8px;
            font-size: 15px;
            font-weight: bold;
            cursor: pointer;
        }

        button:hover {
            background: #0288d1;
            color: white;
        }

        .success {
            color: #00ffae;
            margin-top: 15px;
        }

        .files {
            margin-top: 25px;
            text-align: left;
        }

        .files h3 {
            text-align: center;
            margin-bottom: 10px;
        }

```

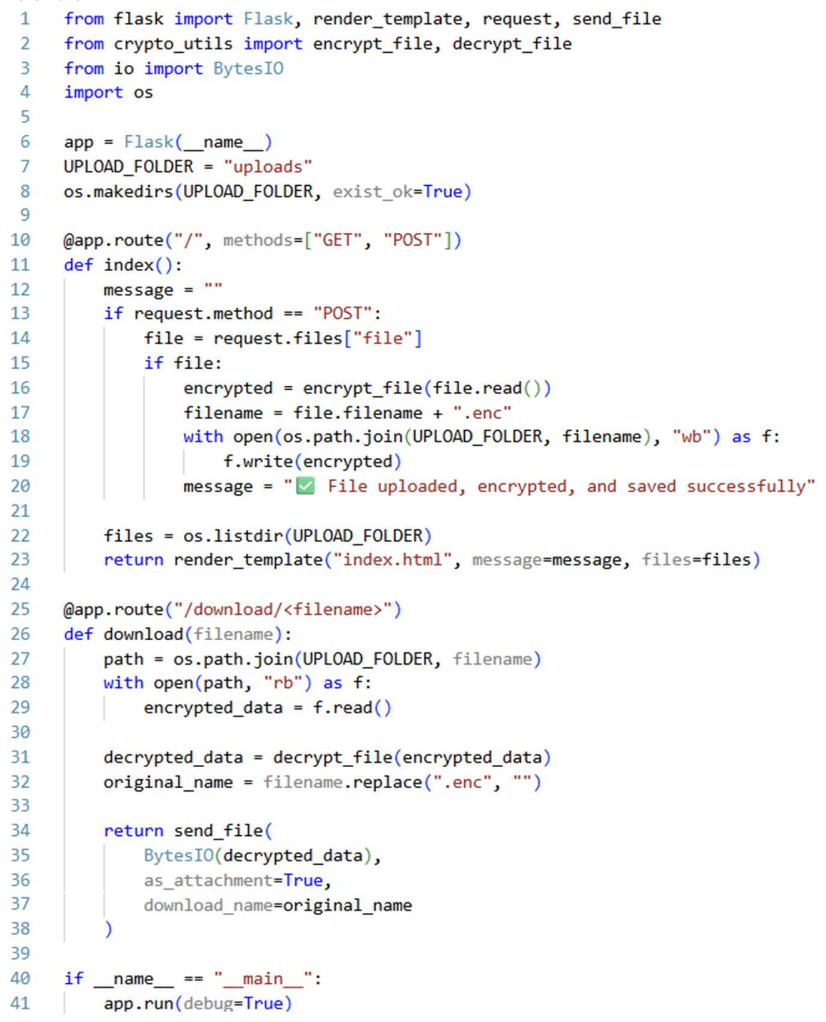


```
        .file-item {
            display: flex;
            justify-content: space-between;
            background: rgba(255,255,255,0.08);
            padding: 8px;
            border-radius: 6px;
            margin-bottom: 6px;
        }

        .file-item a {
            color: #00ffae;
            text-decoration: none;
            font-weight: bold;
        }

        .file-item a:hover {
            text-decoration: underline;
        }
    </style>
</head>

<body>
<div class="container">
    <h1>🔐 Secure File Sharing</h1>

    <form method="POST" enctype="multipart/form-data">
        <input type="file" name="file" required>
        <button type="submit">Encrypt & Upload</button>
    </form>

    {% if message %}
        <p class="success">{{ message }}</p>
```



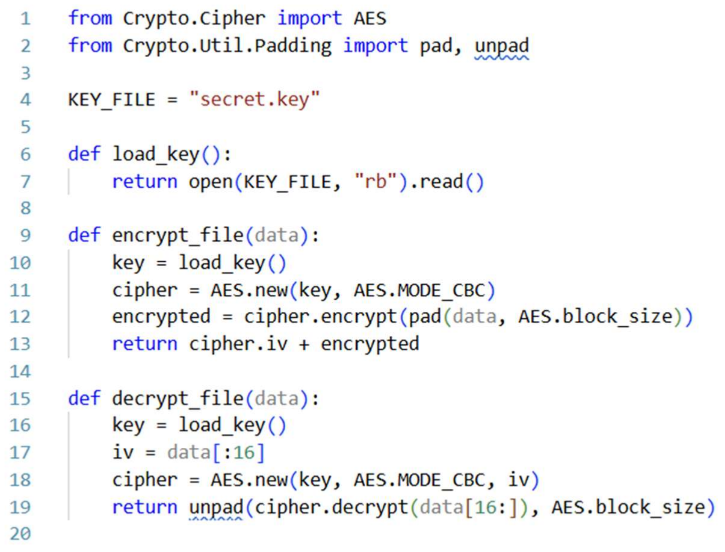
```
    {% endif %}

    <div class="files">
        <h3>📁 Encrypted Files</h3>

        {% for file in files %}
        <div class="file-item">
            <span>{{ file }}</span>
            <a href="/download/{{ file }}">Download</a>
        </div>
        {% endfor %}
    </div>
</div>
</body>
</html>
```

2. App.py



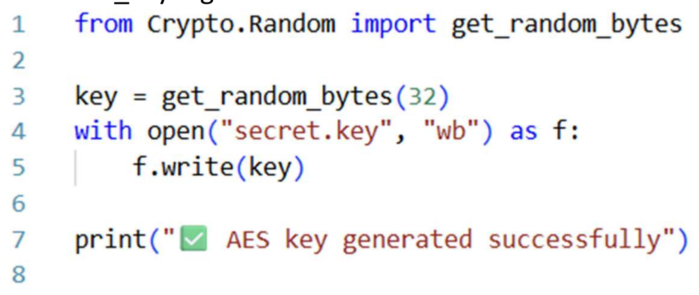
```python
from flask import Flask, render_template, request, send_file
from crypto_utils import encrypt_file, decrypt_file
from io import BytesIO
import os

app = Flask(__name__)
UPLOAD_FOLDER = "uploads"
os.makedirs(UPLOAD_FOLDER, exist_ok=True)

@app.route("/", methods=["GET", "POST"])
def index():
    message = ""
    if request.method == "POST":
        file = request.files["file"]
        if file:
            encrypted = encrypt_file(file.read())
            filename = file.filename + ".enc"
            with open(os.path.join(UPLOAD_FOLDER, filename), "wb") as f:
                f.write(encrypted)
            message = "✅ File uploaded, encrypted, and saved successfully"

    files = os.listdir(UPLOAD_FOLDER)
    return render_template("index.html", message=message, files=files)

@app.route("/download/<filename>")
def download(filename):
    path = os.path.join(UPLOAD_FOLDER, filename)
    with open(path, "rb") as f:
        encrypted_data = f.read()

    decrypted_data = decrypt_file(encrypted_data)
    original_name = filename.replace(".enc", "")

    return send_file(
        BytesIO(decrypted_data),
        as_attachment=True,
        download_name=original_name
    )

if __name__ == "__main__":
    app.run(debug=True)
```

3. Crypto_utils.py



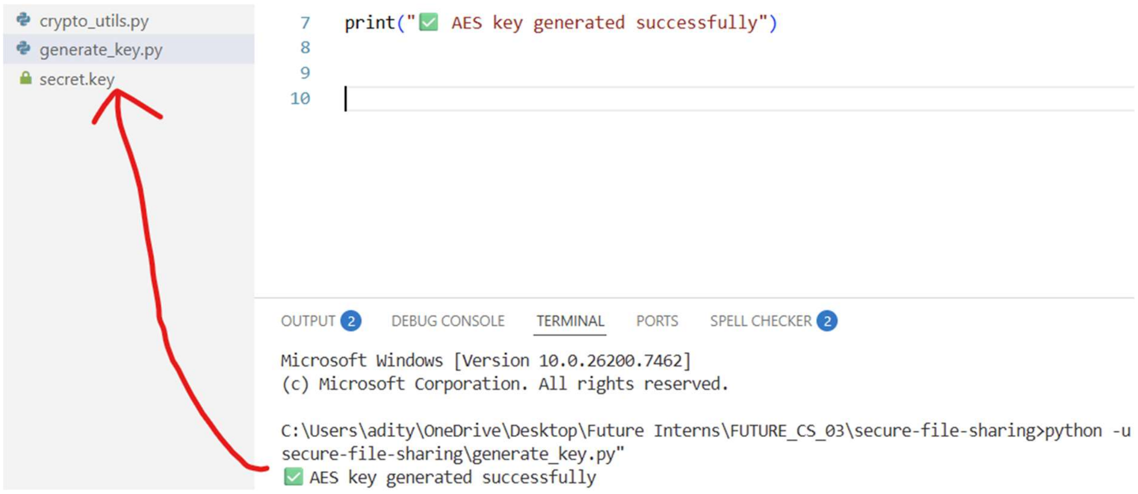
```python
from Crypto.Cipher import AES
from Crypto.Util.Padding import pad, unpad

KEY_FILE = "secret.key"

def load_key():
    return open(KEY_FILE, "rb").read()

def encrypt_file(data):
    key = load_key()
    cipher = AES.new(key, AES.MODE_CBC)
    encrypted = cipher.encrypt(pad(data, AES.block_size))
    return cipher.iv + encrypted

def decrypt_file(data):
    key = load_key()
    iv = data[:16]
    cipher = AES.new(key, AES.MODE_CBC, iv)
    return unpad(cipher.decrypt(data[16:]), AES.block_size)
```

4. Generate_key logic



```python
from Crypto.Random import get_random_bytes

key = get_random_bytes(32)
with open("secret.key", "wb") as f:
    f.write(key)

print("✅ AES key generated successfully")

```

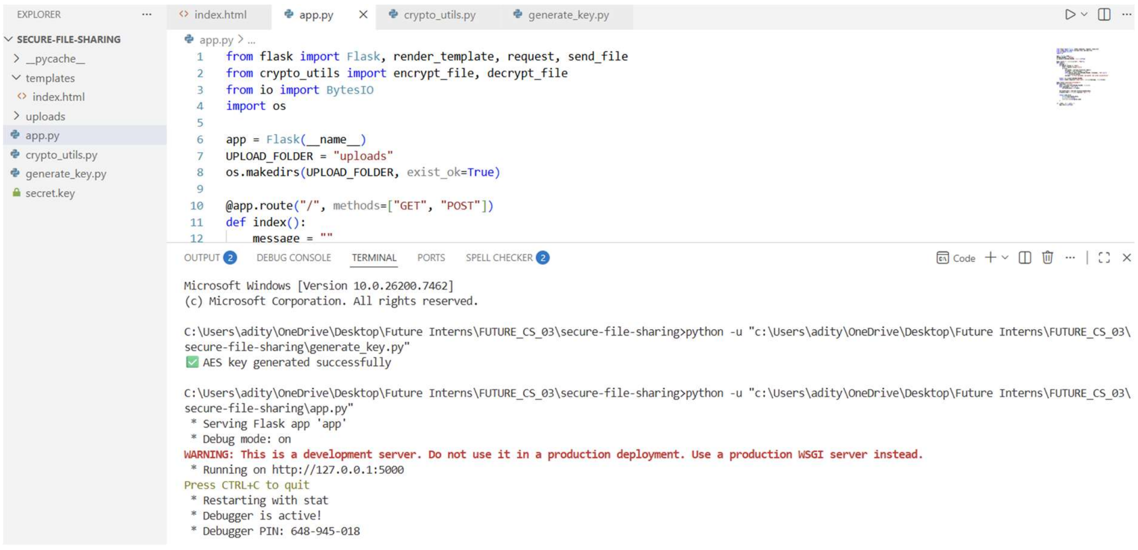
5. Secret key created successfully



```python
print("✅ AES key generated successfully")


```

```
Microsoft Windows [Version 10.0.26200.7462]
(c) Microsoft Corporation. All rights reserved.

C:\Users\adity\OneDrive\Desktop\Future Interns\FUTURE_CS_03\secure-file-sharing>python -u
secure-file-sharing\generate_key.py"
✅ AES key generated successfully
```

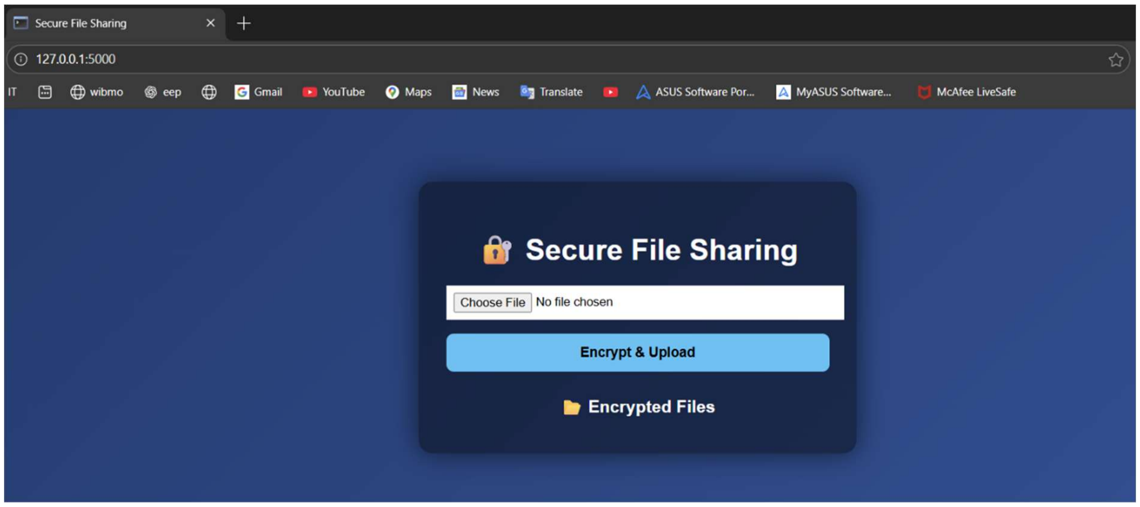
6. Host active and running on 127.0.0.1:5000



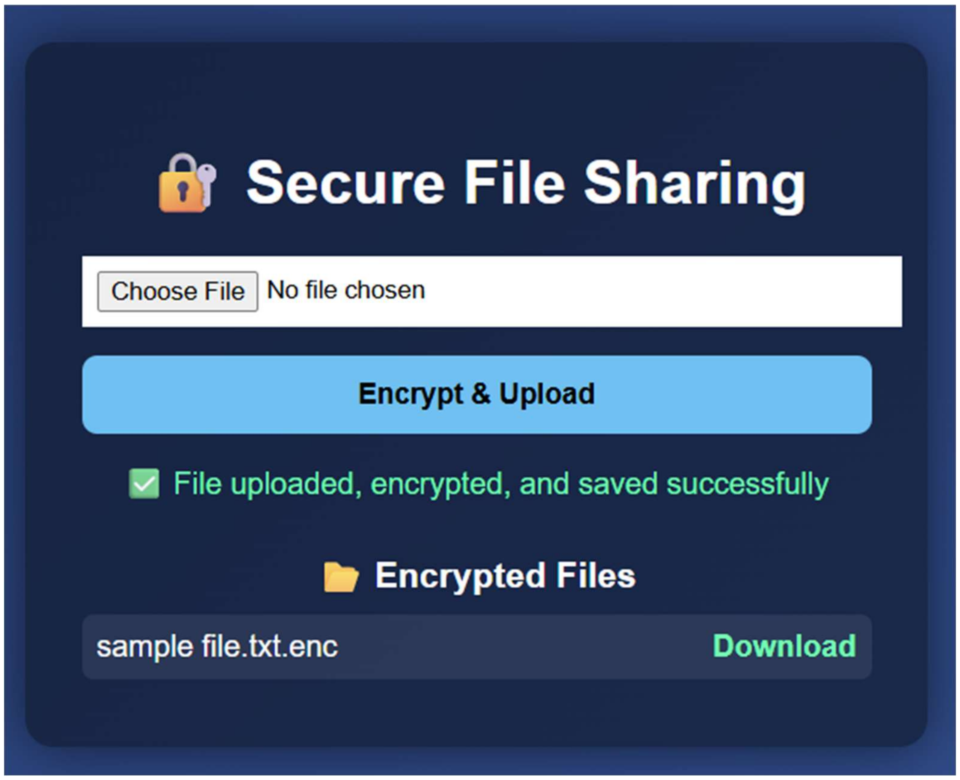
```python
from flask import Flask, render_template, request, send_file
from crypto_utils import encrypt_file, decrypt_file
from io import BytesIO
import os

app = Flask(__name__)
UPLOAD_FOLDER = "uploads"
os.makedirs(UPLOAD_FOLDER, exist_ok=True)

@app.route("/", methods=["GET", "POST"])
def index():
    message = ""
```

```
Microsoft Windows [Version 10.0.26200.7462]
(c) Microsoft Corporation. All rights reserved.

C:\Users\adity\OneDrive\Desktop\Future Interns\FUTURE_CS_03\secure-file-sharing>python -u "c:\Users\adity\OneDrive\Desktop\Future Interns\FUTURE_CS_03\
secure-file-sharing\generate_key.py"
✅ AES key generated successfully

C:\Users\adity\OneDrive\Desktop\Future Interns\FUTURE_CS_03\secure-file-sharing>python -u "c:\Users\adity\OneDrive\Desktop\Future Interns\FUTURE_CS_03\
secure-file-sharing\app.py"
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production 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: 648-945-018
```

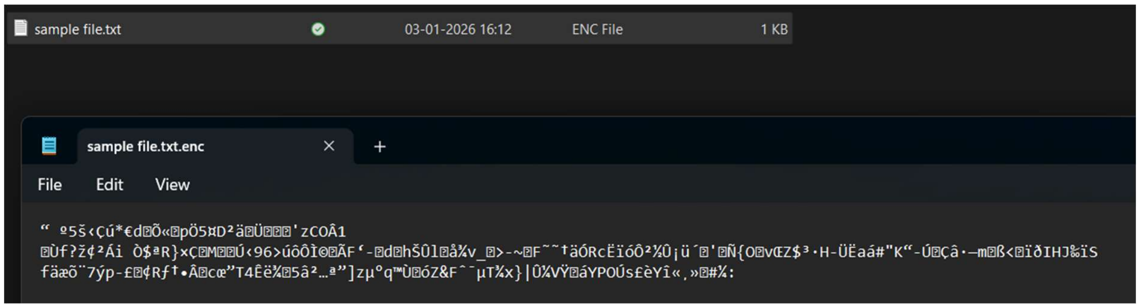
7. User Interface



8. File uploaded and encrypted successfully



9. Encrypted file

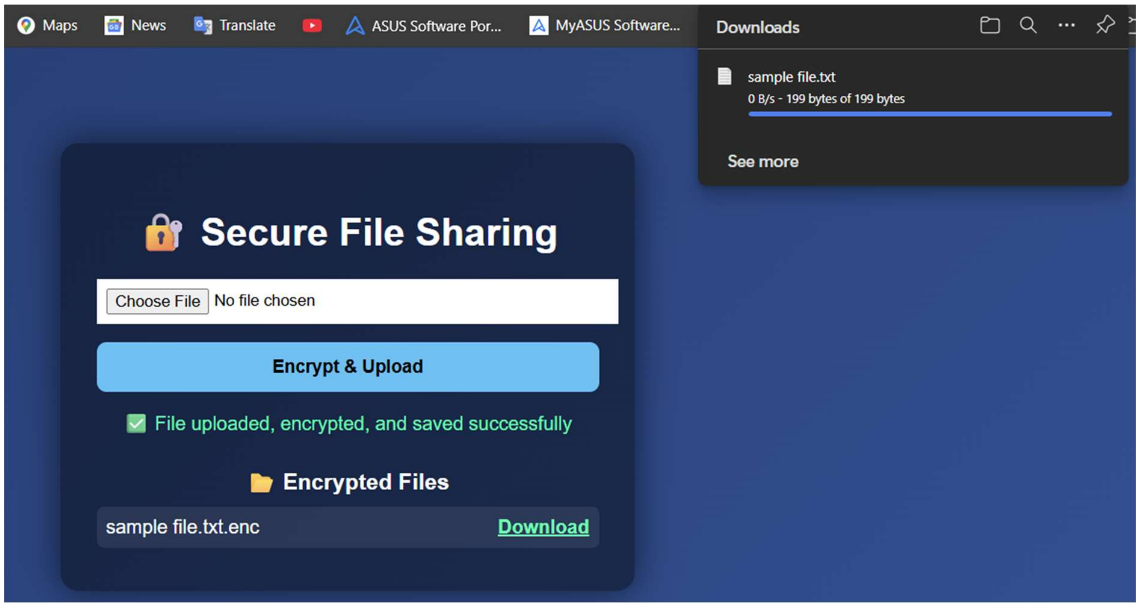


```
sample file.txt    03-01-2026 16:12    ENC File    1 KB
```

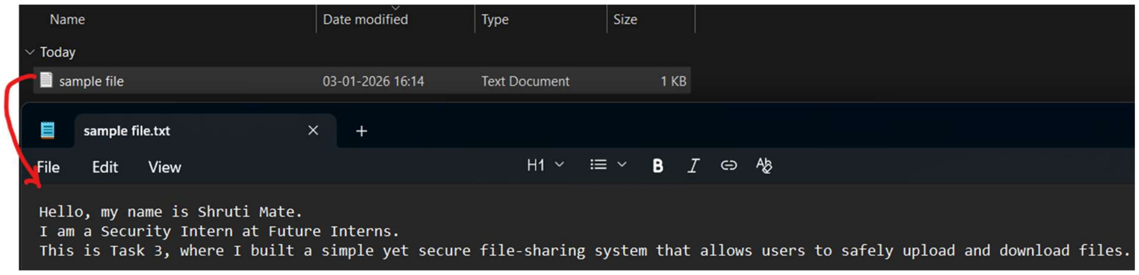
sample file.txt.enc

```
“ º5š‹Çú*€d□Õ«□pÖ5¤D²ä□Ü□□□'zCOÂ1
□Ùf?ž¢²Ái Ò$ªR}×Ç□M□□Ú‹96>úôÔÌ©□ÃF‘-□d□hŠÛl□å¾v_□>-~□F˜˜†äÓRcËïóÔ²½Û¡ü´□'□Ñ{O□vŒZ$³·H-ÜËaá#"K“-Ú□Çâ·—m□ß<□ïðIHJ‰ïS
fäæõ¨7ýp-£□¢Rƒ†•Â□cœ”T4Êë¾□5â²…ª”]zµ°q™Ù□óZ&F˜¯µT¾x}|Û½VŸ□áYPOÚs£èYî«‚»□#¼:
```

10. Downloading decrypted file



11. Decrypted file

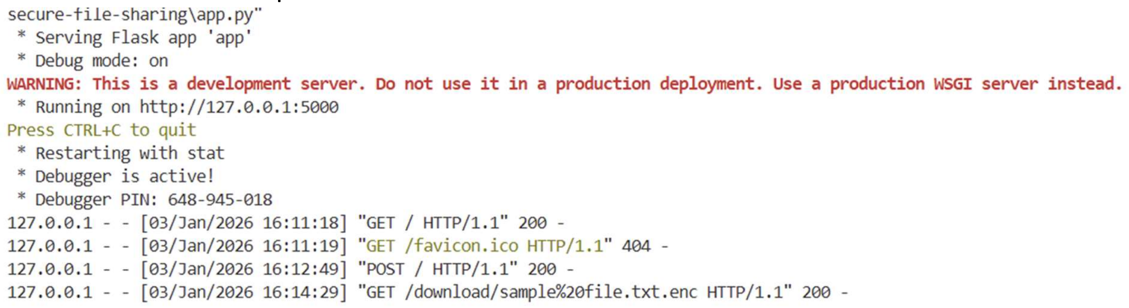


Name	Date modified	Type	Size
Today			
sample file	03-01-2026 16:14	Text Document	1 KB

sample file.txt

```
Hello, my name is Shruti Mate.
I am a Security Intern at Future Interns.
This is Task 3, where I built a simple yet secure file-sharing system that allows users to safely upload and download files.
```

12. GET and POST with response time.



```
secure-file-sharing\app.py"
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production 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: 648-945-018
127.0.0.1 - - [03/Jan/2026 16:11:18] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [03/Jan/2026 16:11:19] "GET /favicon.ico HTTP/1.1" 404 -
127.0.0.1 - - [03/Jan/2026 16:12:49] "POST / HTTP/1.1" 200 -
127.0.0.1 - - [03/Jan/2026 16:14:29] "GET /download/sample%20file.txt.enc HTTP/1.1" 200 -
```