TASK 3 : SECURE FILE SHARING SYSTEM

Intern Name: Praveen R

Task Code: FUTURE_CS_03

Internship Program: Cybersecurity Internship by Future intern

Report Title: Secure File Sharing System

Date Submitted: 08 / 07 / 2025

Aim:

The aim of this project is to build a simple, secure file-sharing system that allows users to upload, download, modify, and delete files while ensuring the confidentiality and integrity of the files. The system utilizes **AES encryption** to encrypt files before storage and decryption during download. The project simulates real-world environments where secure data sharing is critical, such as healthcare, legal, and corporate fields.

Introduction:

The system allows users to interact with a web portal that enables file uploads, downloads, modification, and deletion. The application ensures security through the use of AES encryption for file storage and retrieval, providing protection against unauthorized access. It also offers a simple and intuitive user interface, making it easy to upload and retrieve files while maintaining security best practices.

Technology Stack

Frontend: HTML, CSS, JavaScript

Backend: Python Flask (API endpoints for file handling)

Encryption: AES (Advanced Encryption Standard)

Command Line: `curl` for testing the system via API calls

Code Explanation:

Backend (app.py)

The backend of the system is implemented using **Python Flask**. Here's a breakdown of its functionality:

Flask Setup:
The application is set up using the Flask framework with basic routing to handle various requests like file upload, download, modification, and deletion.

AES Encryption:
The AES encryption is performed using the `pycryptodome` library. The `encrypt_file` function takes the file data and encrypts it using a randomly generated 32-byte key in CBC mode. The IV (Initialization Vector) is also generated randomly to ensure that each encryption is unique.

File Upload:
When a user uploads a file, it is encrypted and stored in the `uploads` folder with a unique filename (UUID-based). The encryption key and file hash are stored in the `KEY_STORE` dictionary to allow decryption when the file is downloaded.

File Download:
When a user requests to download a file, the system checks if the file exists. If the file is found, it is decrypted using the stored key, and the integrity of the file is verified by comparing its hash with the stored hash.

File Modification:
Users can replace a file by uploading a new version. The file is re-encrypted, and the key and hash are updated in the `KEY_STORE`.

File Deletion:
Users can delete a file, which removes the encrypted file from storage and also updates the `KEY_STORE`.

```python
from flask import Flask, request, jsonify, send_file, render_template
from Crypto.Cipher import AES
from Crypto.Random import get_random_bytes
from Crypto.Util.Padding import pad, unpad
import hashlib
import os
from io import BytesIO
import uuid

app = Flask(__name__)
UPLOAD_FOLDER = 'uploads'
os.makedirs(UPLOAD_FOLDER, exist_ok=True)
BLOCK_SIZE = 16
KEY_STORE = {}  # stored_name -> {key, hash, original}
NAME_MAP = {}   # original_name -> stored_name (UUID prefixed)
def get_sha256(data):
    return hashlib.sha256(data).hexdigest()
def encrypt_file(data, key):
    iv = get_random_bytes(16)
    cipher = AES.new(key, AES.MODE_CBC, iv)
    return iv + cipher.encrypt(pad(data, BLOCK_SIZE))
def decrypt_file(data, key):
    iv = data[:16]
    ciphertext = data[16:]
    cipher = AES.new(key, AES.MODE_CBC, iv)
    return unpad(cipher.decrypt(ciphertext), BLOCK_SIZE)
@app.route('/')
def index():
    return render_template('index.html')
@app.route('/upload', methods=['POST'])
def upload():
    if 'file' not in request.files:
        return jsonify({"error": "No file part"}), 400
    file = request.files['file']
    if file.filename == '':
        return jsonify({"error": "No selected file"}), 400
    original_name = file.filename
    data = file.read()
    key = get_random_bytes(32)
    encrypted = encrypt_file(data, key)
    file_hash = get_sha256(data)
    # Save with UUID prefix
    stored_name = f"{uuid.uuid4().hex}_{original_name}"
    path = os.path.join(UPLOAD_FOLDER, stored_name)
    with open(path, 'wb') as f:
        f.write(encrypted)
    # Store key and mapping
    KEY_STORE[stored_name] = {'key': key, 'hash': file_hash, 'original': original_name}
    NAME_MAP[original_name] = stored_name
    return jsonify({"message": "File uploaded successfully.", "filename": original_name}), 200
@app.route('/download/<filename>', methods=['GET'])
def download(filename):
    stored_name = NAME_MAP.get(filename)
    if not stored_name:
        return jsonify({"error": "File not found"}), 404
    path = os.path.join(UPLOAD_FOLDER, stored_name)
    try:
        with open(path, 'rb') as f:
            encrypted = f.read()
        key_data = KEY_STORE.get(stored_name)
        if not key_data:
            return jsonify({"error": "Encryption key missing"}), 403
        decrypted = decrypt_file(encrypted, key_data['key'])
        if get_sha256(decrypted) != key_data['hash']:
            return jsonify({"error": "File integrity compromised"}), 400
        return send_file(BytesIO(decrypted), as_attachment=True,
download_name=key_data['original'])
    except Exception as e:
        return jsonify({"error": str(e)}), 500
@app.route('/files', methods=['GET'])
def list_files():
    return jsonify({"files": list(NAME_MAP.keys())}), 200
@app.route('/delete/<filename>', methods=['DELETE'])
def delete(filename):
    stored_name = NAME_MAP.get(filename)
    if not stored_name:
        return jsonify({"error": "File not found"}), 404
    path = os.path.join(UPLOAD_FOLDER, stored_name)
    if os.path.exists(path):
        os.remove(path)
    KEY_STORE.pop(stored_name, None)
    NAME_MAP.pop(filename, None)
    return jsonify({"message": "File deleted"}), 200
@app.route('/modify/<filename>', methods=['PUT'])
def modify(filename):
    stored_name = NAME_MAP.get(filename)
    if not stored_name:
        return jsonify({"error": "Original file not found"}), 404
    if 'file' not in request.files:
        return jsonify({"error": "No file uploaded"}), 400
    new_file = request.files['file']
    data = new_file.read()
    key = get_random_bytes(32)
    encrypted = encrypt_file(data, key)
    file_hash = get_sha256(data)
    path = os.path.join(UPLOAD_FOLDER, stored_name)
    with open(path, 'wb') as f:
        f.write(encrypted)
    KEY_STORE[stored_name] = {'key': key, 'hash': file_hash, 'original': filename}
    return jsonify({"message": "File modified and re-encrypted"}), 200
if __name__ == '__main__':
    app.run(debug=True)
```

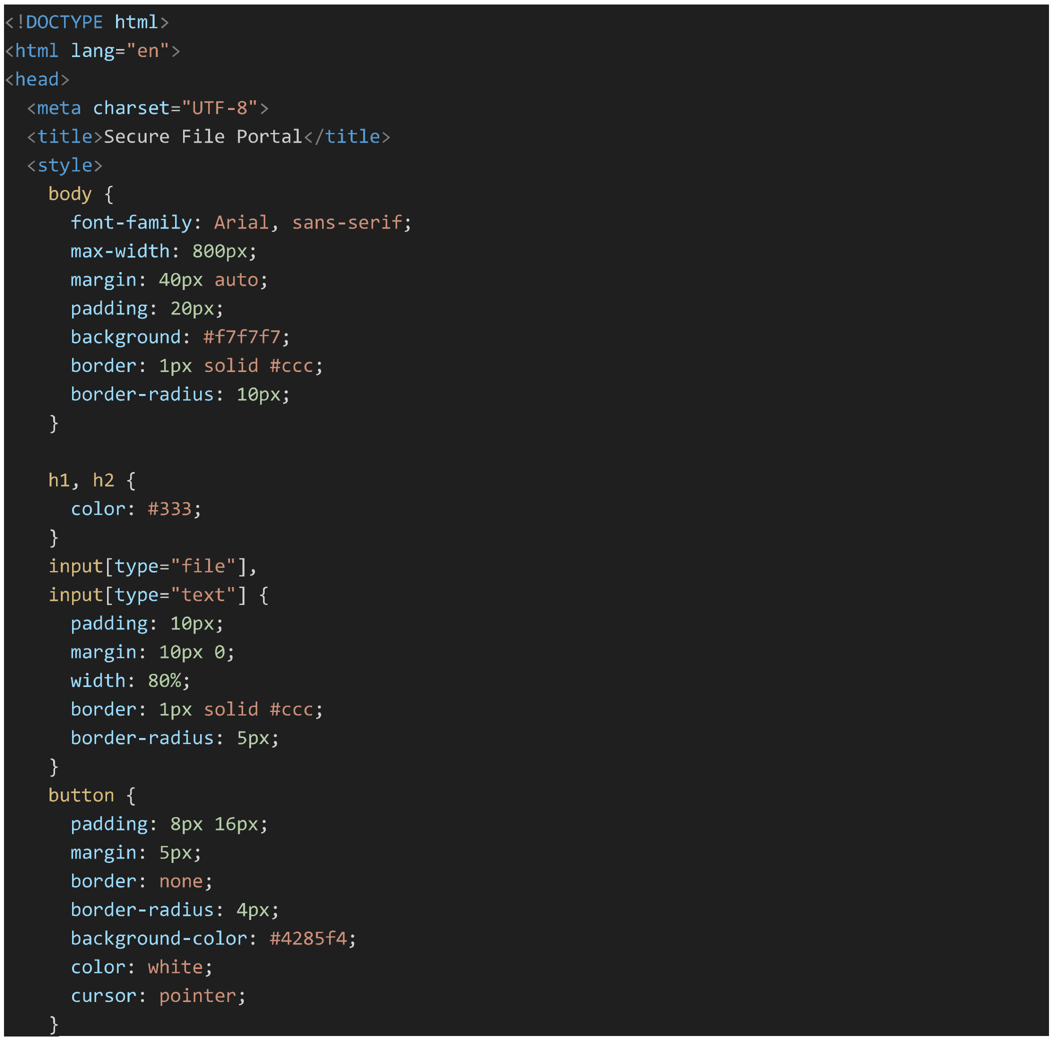
Frontend (HTML, CSS, and JavaScript)

HTML:
The HTML provides the basic structure for the file-sharing portal. It includes forms for file upload, an input box for file search, and buttons for interacting with the files (download, delete, modify).

CSS (Integrated in HTML):
The **CSS** styles are embedded directly within the `<style>` tag in the HTML file. This provides a clean and simple layout for the application. The interface is designed to be user-friendly, with input fields, buttons, and file lists styled for ease of use.

Key CSS elements:

> **Body Styling**: The body is styled to have a light gray background, centered content with a max-width for readability, and some padding for spacing.
>
> **Buttons**: Buttons are styled with padding, background color, and hover effects to improve user interaction.
>
> **File List**: Files are displayed in a neat list with each file in a box containing buttons for downloading, modifying, or deleting.

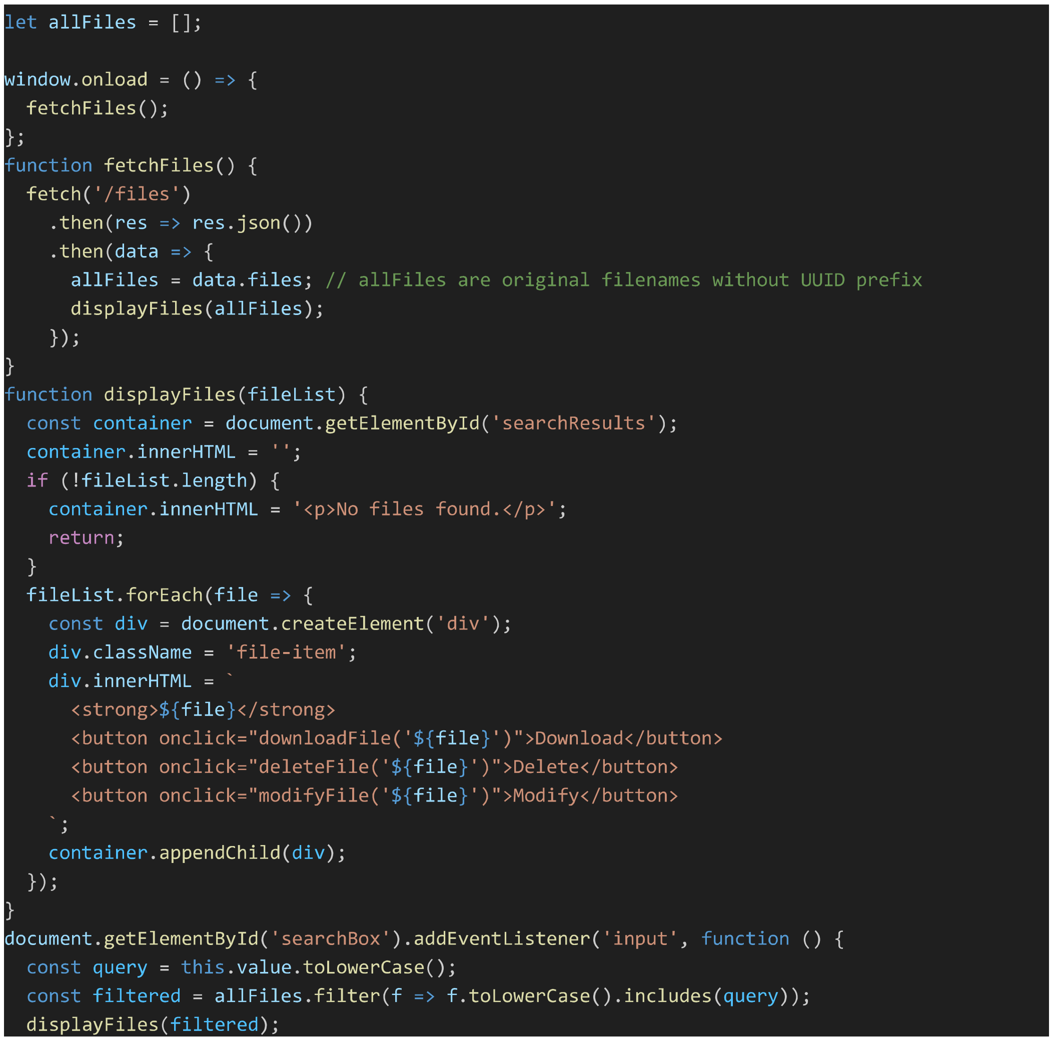


```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Secure File Portal</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      max-width: 800px;
      margin: 40px auto;
      padding: 20px;
      background: #f7f7f7;
      border: 1px solid #ccc;
      border-radius: 10px;
    }

    h1, h2 {
      color: #333;
    }
    input[type="file"],
    input[type="text"] {
      padding: 10px;
      margin: 10px 0;
      width: 80%;
      border: 1px solid #ccc;
      border-radius: 5px;
    }
    button {
      padding: 8px 16px;
      margin: 5px;
      border: none;
      border-radius: 4px;
      background-color: #4285f4;
      color: white;
      cursor: pointer;
    }
    button:hover {
      background-color: #357ae8;
    }
    .file-list {
      margin-top: 30px;
    }
    .file-item {
      background: #fff;
      padding: 10px;
      margin: 5px 0;
      border: 1px solid #ccc;
      border-radius: 5px;
    }
    .file-item button {
      margin-left: 10px;
    }
  </style>
</head>
<body>
  <h1>Secure File Portal</h1>
  <!-- Upload -->
  <h2>Upload File</h2>
  <form id="uploadForm">
    <input type="file" id="uploadFile" name="file" required />
    <button type="submit">Upload</button>
  </form>
  <!-- Search -->
  <h2>Search & Download File</h2>
  <input type="text" id="searchBox" placeholder="Type part of filename..." />
  <div id="searchResults" class="file-list"></div>
  <script>
    let allFiles = [];
    // Load on start
    window.onload = () => {
      fetchFiles();
    };
    function fetchFiles() {
      fetch('/files')
        .then(res => res.json())
        .then(data => {
          allFiles = data.files;
          displayFiles(data.files);
        });
    }
    function displayFiles(fileList) {
      const container = document.getElementById('searchResults');
      container.innerHTML = '';
      if (!fileList.length) {
        container.innerHTML = '<p>No files found.</p>';
        return;
      }
      fileList.forEach(file => {
        const div = document.createElement('div');
        div.className = 'file-item';
        div.innerHTML = `
          <strong>${file}</strong>
          <button onclick="downloadFile('${file}')">Download</button>
          <button onclick="deleteFile('${file}')">Delete</button>
          <button onclick="modifyFile('${file}')">Modify</button>
        `;
        container.appendChild(div);
      });
    }
    document.getElementById('searchBox').addEventListener('input', function () {
      const query = this.value.toLowerCase();
      const filtered = allFiles.filter(f => f.toLowerCase().includes(query));
      displayFiles(filtered);
    });
    document.getElementById('uploadForm').addEventListener('submit', function (e) {
      e.preventDefault();
      const fileInput = document.getElementById('uploadFile');
      const file = fileInput.files[0];
      if (!file) return alert("Select a file first.");
      const formData = new FormData();
      formData.append('file', file);
      fetch('/upload', {
        method: 'POST',
        body: formData
      })
        .then(res => res.json())
        .then(data => {
          alert(data.message || data.error);
          fileInput.value = '';
          fetchFiles();
        });
    });
    function downloadFile(filename) {
      window.location.href = '/download/' + encodeURIComponent(filename);
    }
    function deleteFile(filename) {
      if (!confirm(`Delete ${filename}?`)) return;
      fetch('/delete/' + encodeURIComponent(filename), {
        method: 'DELETE'
      })
        .then(res => res.json())
        .then(data => {
          alert(data.message || data.error);
          fetchFiles();
        });
    }
    function modifyFile(filename) {
      const fileInput = document.createElement('input');
      fileInput.type = 'file';
      fileInput.accept = '*/*';
      fileInput.onchange = function () {
        const formData = new FormData();
        formData.append('file', fileInput.files[0]);
        fetch('/modify/' + encodeURIComponent(filename), {
          method: 'PUT',
          body: formData
        })
          .then(res => res.json())
          .then(data => {
            alert(data.message || data.error);
            fetchFiles();
          });
      };
      fileInput.click();
    }
  </script>
</body>
</html>
```

JavaScript:
JavaScript is used to handle dynamic content and interaction with the backend via the API. It performs tasks like:

> **Uploading Files**: It handles the form submission and sends the file to the `/upload` API using `fetch`.
>
> **Displaying Files**: It fetches the list of available files from the `/files` API and displays them in a user-friendly list.
>
> **Searching Files**: The search functionality dynamically filters the list of files as the user types in the search box.
>
> **File Operations**: JavaScript also handles file downloads, deletions, and modifications by calling the respective APIs via `fetch`.



```javascript
let allFiles = [];

window.onload = () => {
  fetchFiles();
};
function fetchFiles() {
  fetch('/files')
    .then(res => res.json())
    .then(data => {
      allFiles = data.files; // allFiles are original filenames without UUID prefix
      displayFiles(allFiles);
    });
}
function displayFiles(fileList) {
  const container = document.getElementById('searchResults');
  container.innerHTML = '';
  if (!fileList.length) {
    container.innerHTML = '<p>No files found.</p>';
    return;
  }
  fileList.forEach(file => {
    const div = document.createElement('div');
    div.className = 'file-item';
    div.innerHTML = `
      <strong>${file}</strong>
      <button onclick="downloadFile('${file}')">Download</button>
      <button onclick="deleteFile('${file}')">Delete</button>
      <button onclick="modifyFile('${file}')">Modify</button>
    `;
    container.appendChild(div);
  });
}
document.getElementById('searchBox').addEventListener('input', function () {
  const query = this.value.toLowerCase();
  const filtered = allFiles.filter(f => f.toLowerCase().includes(query));
  displayFiles(filtered);
});
document.getElementById('uploadForm').addEventListener('submit', function (e) {
  e.preventDefault();
  const fileInput = document.getElementById('uploadFile');
  const file = fileInput.files[0];
  if (!file) return alert("Please select a file to upload.");
  const formData = new FormData();
  formData.append('file', file);
  fetch('/upload', {
    method: 'POST',
    body: formData
  })
    .then(res => res.json())
    .then(data => {
      alert(data.message || data.error);
      fileInput.value = '';
      fetchFiles();
    });
});
function downloadFile(filename) {
  // Filename is original name, backend maps it internally
  window.location.href = '/download/' + encodeURIComponent(filename);
}
function deleteFile(filename) {
  if (!confirm(`Are you sure you want to delete "${filename}"?`)) return;
  fetch('/delete/' + encodeURIComponent(filename), {
    method: 'DELETE'
  })
    .then(res => res.json())
    .then(data => {
      alert(data.message || data.error);
      fetchFiles();
    });
}
function modifyFile(filename) {
  const fileInput = document.createElement('input');
  fileInput.type = 'file';
  fileInput.accept = '*/*';
  fileInput.onchange = function () {
    const formData = new FormData();
    formData.append('file', fileInput.files[0]);
    fetch('/modify/' + encodeURIComponent(filename), {
      method: 'PUT',
      body: formData
    })
      .then(res => res.json())
      .then(data => {
        alert(data.message || data.error);
        fetchFiles();
      });
  };
  fileInput.click();
}
```

Web Interface and Curl Commands Explanation

Interacting with the Backend via `curl`

You can interact with the backend system directly from the command line using `curl` to perform file operations. Below are examples for testing each API endpoint:

1. File Upload

To upload a file using `curl`, run the following command:

```
curl -X POST -F "file=@/path/to/your/file.txt"
http://127.0.0.1:5000/upload
```

Example
Uploading `Metrics (1).pdf` from `E:\Cyber\`:

```
curl -X POST -F "file=@E:\Cyber\Metrics (1).pdf"
http://127.0.0.1:5000/upload
```

Response:

```
{
  "filename": "Metrics (1).pdf",
  "message": "File uploaded successfully."}
```

2. List Uploaded Files

To retrieve a list of all uploaded files:

```
curl http://127.0.0.1:5000/files
```

Response:

```
{
  "files": [
    "Metrics (1).pdf"
  ]}
```

3. File Download

To download a specific file:

```
curl -O -J "http://127.0.0.1:5000/download/<filename>"
```

Example:

```
curl -O -J "http://127.0.0.1:5000/download/Metrics%20(1).pdf"
```

Alternatively, to specify the output path:

```
curl -J "http://127.0.0.1:5000/download/Metrics%20(1).pdf" -o
"E:\Cyber\Downloaded_Metrics.pdf"
```

4. File Deletion

To delete an uploaded file:

```
curl -X DELETE http://127.0.0.1:5000/delete/<filename>
```

Example:

```
curl -X DELETE http://127.0.0.1:5000/delete/Metrics%20(1).pdf
```

Response:

```
{
  "message": "File deleted"
}
```

5. File Modification (Replace Existing File)

To replace the contents of an existing file:

```
curl -X PUT -F "file=@/path/to/new/file.txt"
http://127.0.0.1:5000/modify/<filename>
```

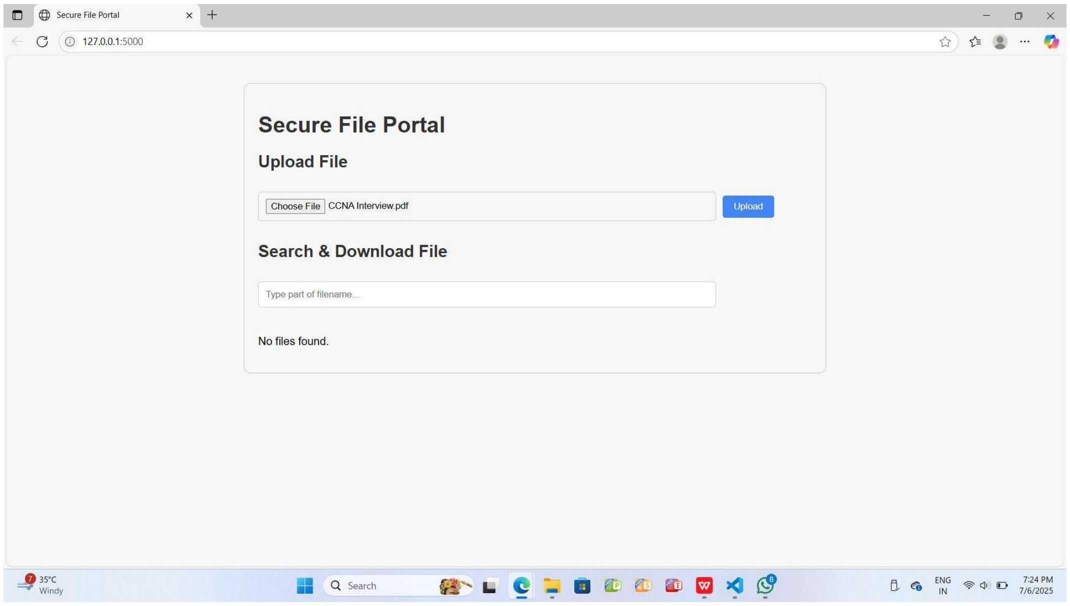
Example:

```
curl -X PUT -F "file=@E:\Cyber\Updated_Metrics.pdf"
http://127.0.0.1:5000/modify/Metrics%20(1).pdf
```

Example Screenshots

Below are the screenshots showcasing various features of the Secure File Portal:

This is my website interface



Secure File Portal

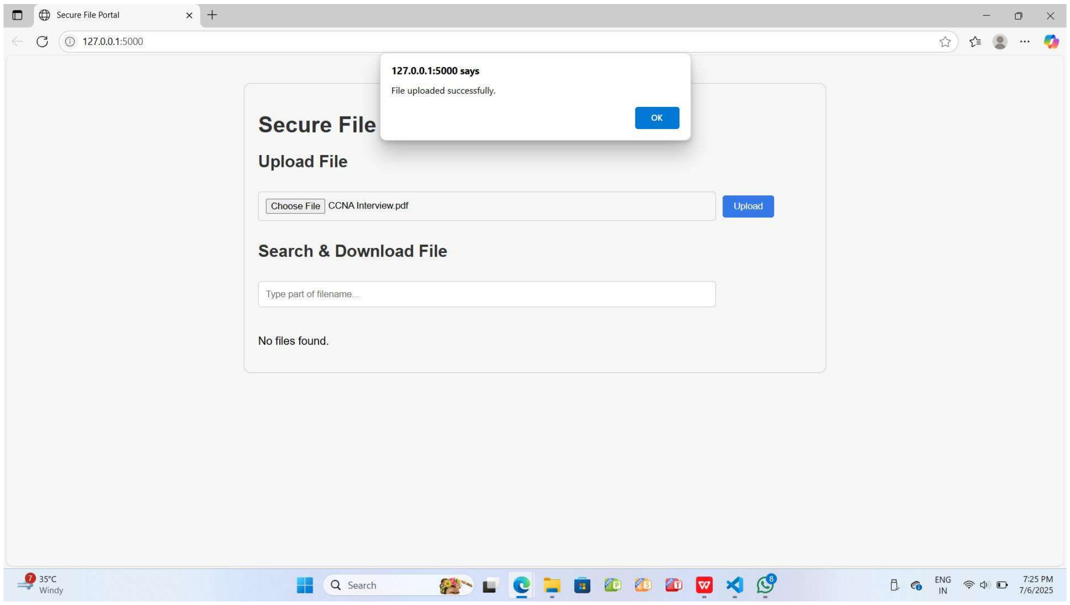
Upload File

CCNA Interview.pdf

Search & Download File

No files found.

I have uploaded the CCNA Interview.pdf by clicking the ‘Choose File’ button.



127.0.0.1:5000 says

File uploaded successfully.

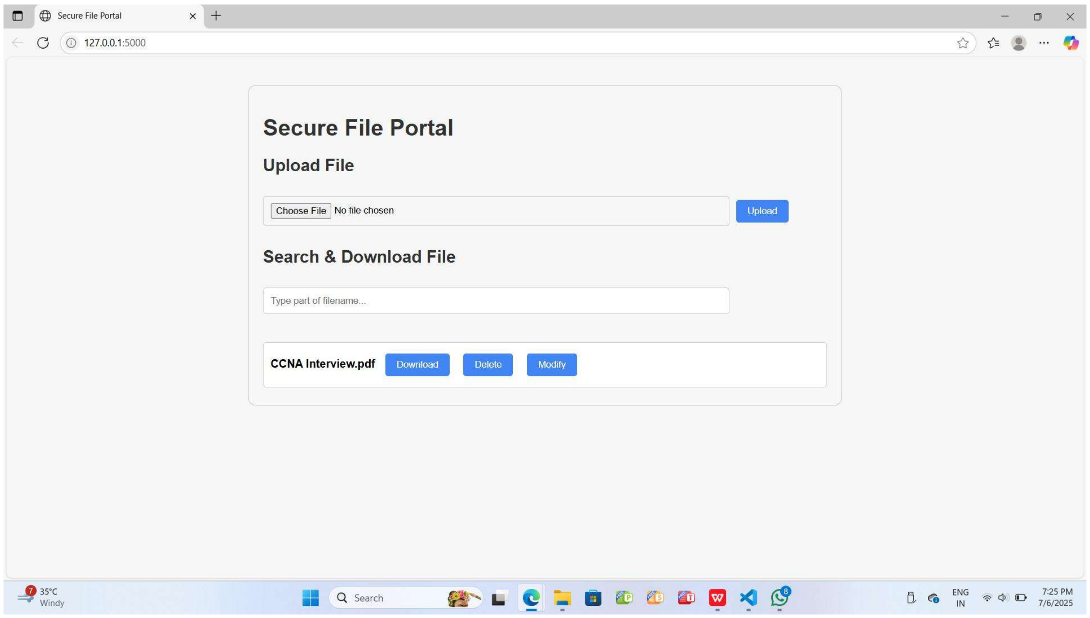
Secure File Portal

Upload File

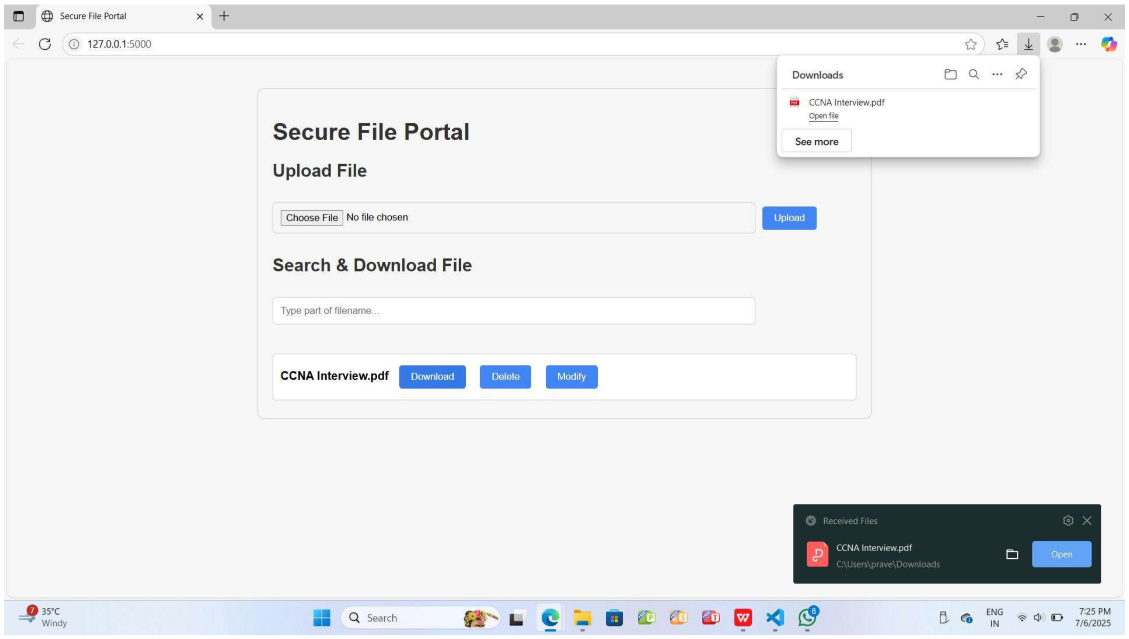
CCNA Interview.pdf

Search & Download File

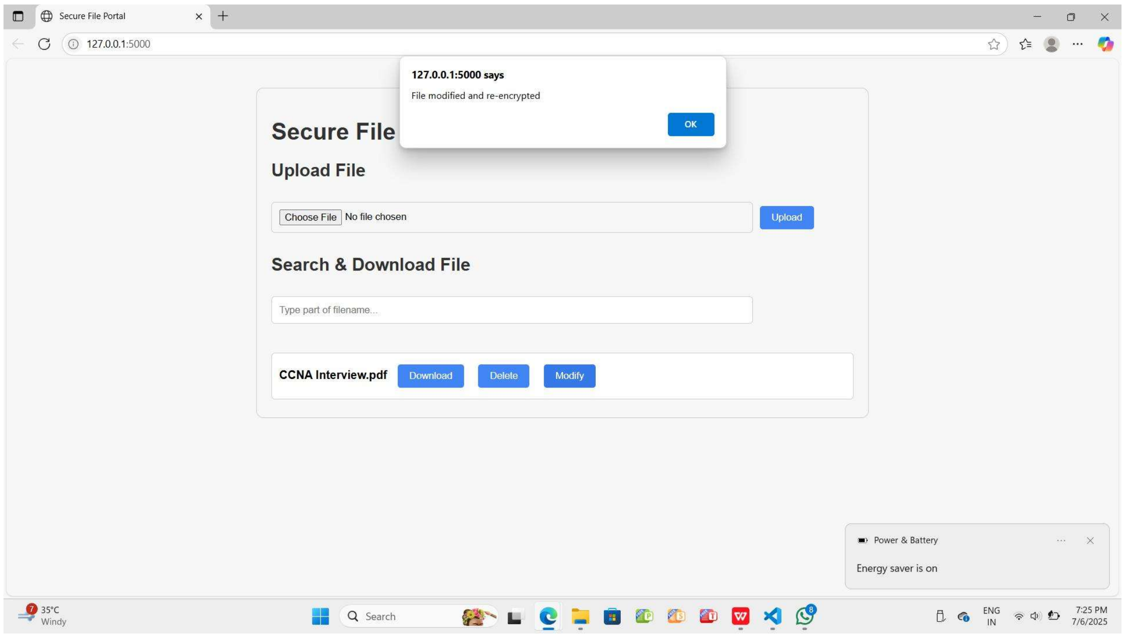
No files found.



I downloaded the *CCNA Interview.pdf* file by clicking the download button located at the bottom of the search section. Alternatively, we can use the search section to download the file by searching for its name.



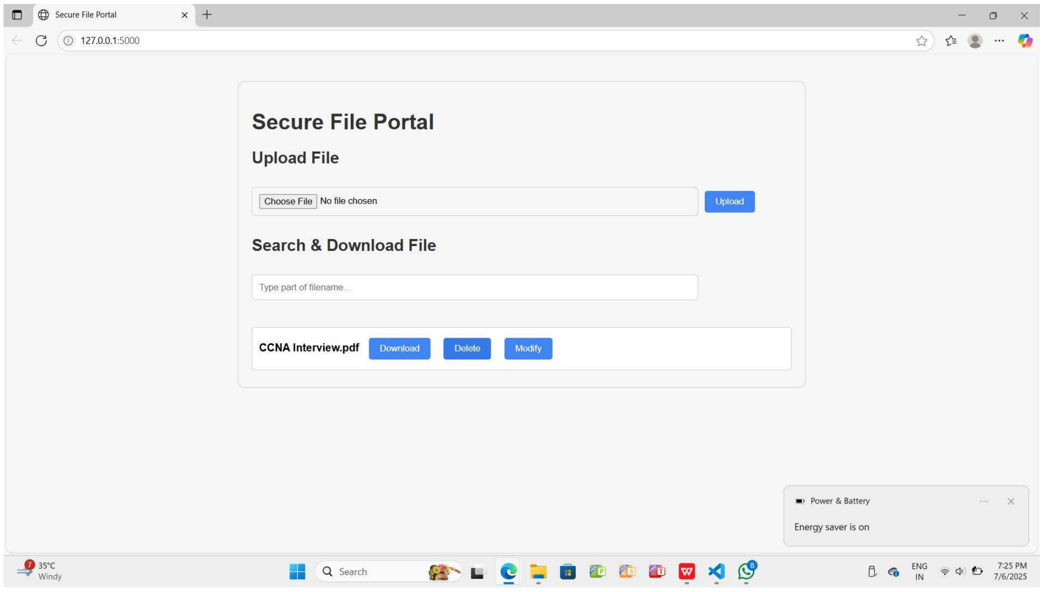
I clicked the Modify button and reuploaded the file. The Modify button is used to re-encrypt the file with a new key



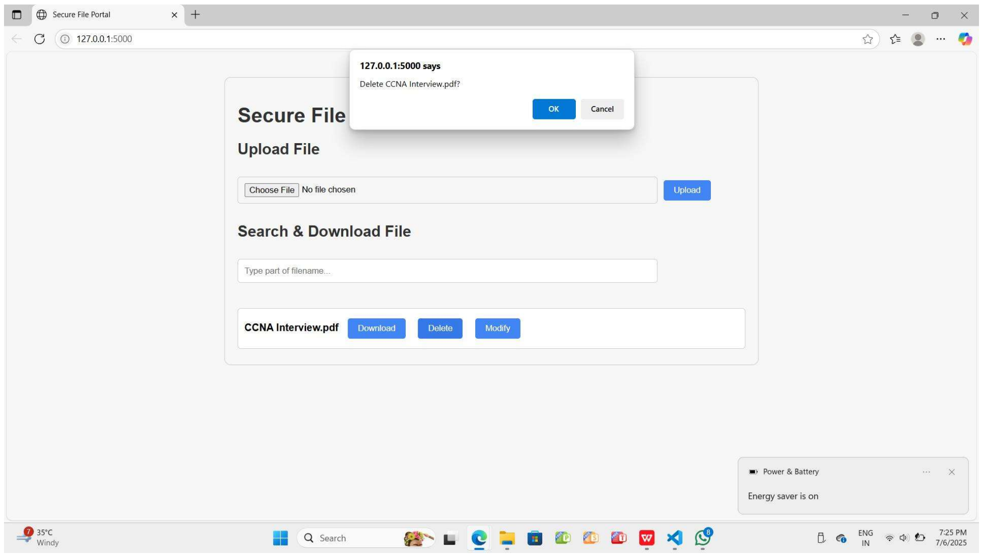
127.0.0.1:5000 says

File modified and re-encrypted

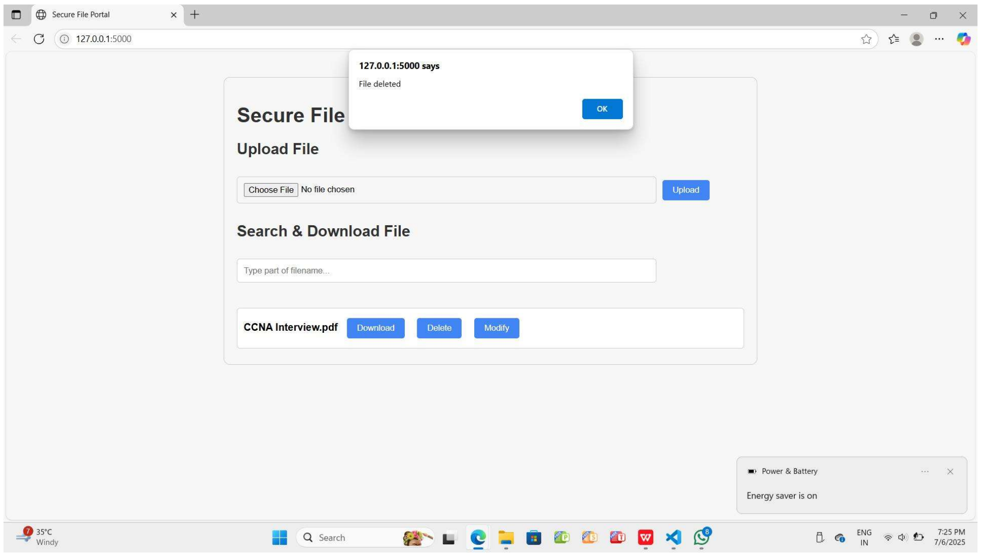
The image below shows the re-encrypted file, which is the modified PDF.



By clicking the Delete button, the file stored on the website is deleted.


127.0.0.1:5000 says

Delete CCNA Interview.pdf?


127.0.0.1:5000 says

File deleted

Conclusion:

This secure file-sharing system ensures that sensitive files can be uploaded, stored, and retrieved with robust encryption (AES). By utilizing Python Flask for the backend and a simple HTML/JS interface, the system provides a seamless experience for users while ensuring data integrity and confidentiality. The added feature of command-line interaction using `curl` allows for easy integration and testing of the system's API endpoints.