Cryptography and Network Security

File Hash Calculator

Core Project Structure

1. **Project Setup:**
    - Create a project folder.
    - Initialize a virtual environment (recommended).
    - Install Flask: `pip install Flask`
    - Create subfolders:
        - `templates` (for your HTML files)
        - `uploads` (to store encrypted/decrypted files)
2. **Base Python File (`app.py`):**
    - **Imports:**

      Python
      ```
      from flask import Flask, request, render_template,
      send_from_directory, url_for
      import os
      import hashlib
      from cryptography.fernet import Fernet
      from werkzeug.utils import secure_filename
      import shutil
      import time
      ```
    - **Flask App:**

      Python
      ```
      app = Flask(__name__, template_folder='templates')
      app.config['UPLOAD_FOLDER'] = 'uploads'
      ```
    - **Global Lists (for data storage):**

      Python
      ```
      file_sizes = []
      encryption_times = []
      decryption_times = []
      ```

Functionality

3. **Encryption:**
    - `/` **Route (Homepage):**
        - HTML form for file upload (`method="POST"`)
        - Endpoint handling in `app.py`:
            - Secure file retrieval from the form
            - Generate an encryption key (Fernet)
            - Encrypt the file
            - Store the encrypted file in the 'uploads' folder
            - Calculate file size and encryption time
            - Store data in the `file_sizes` and `encryption_times` lists
            - Render `index.html` with success messages and encryption details
4. **Decryption**
    - `/decrypt` **Route:**
        - HTML form for:
            - Uploading the encrypted file
            - Entering the original file hash (for comparison)
            - Entering the encryption key
        - Endpoint handling in `app.py`:
            - Retrieve form data
            - Decrypt the file
            - Calculate file size and decryption time
            - Store data in the `file_sizes` and `decryption_times` lists
            - Calculate the new hash of the decrypted file
            - Compare hashes for validation
            - Render `index.html` with decryption results
5. **HTML Template (`index.html`)**
    - Forms for encryption and decryption
    - Dynamic display of results:
        - Original file hash, encryption key, file path, etc.

    - Decryption status, new file hash
    - Encryption and decryption times

Additional Considerations

- **Error Handling:** Implement `try...except` blocks for robust error handling.
- **Hash Storage:** Consider how you'll store the original file hashes (text file, simple database).
- **Security:** Explore additional security measures if handling sensitive data.