High-Level Design of the Proposed Solution

1. System Overview

We propose to develop a secure file processing **web application** that allows users to protect sensitive files from unauthorized access, alterations, or forgery. It does so by offering:

- **Encryption and Decryption—** to protect the file's confidentiality.

- **Hashing—**to verify a file's integrity.

The resulting files (hash/encrypted files) will be saved and made available for download. The app will provide a **simple and secure interface** to make file security accessible.

2. System Design & Architecture

2.1 Architecture Type

Our web application is built using a **three-layer architecture**, which makes the system easier to maintain, more secure, and well-organized:

- **Presentation Layer:**

This is the part the user interacts with. It includes a simple web interface created using **Django Templates** (HTML, CSS, JavaScript), where the user can upload a file, choose an operation (like encryption, decryption, or hashing), and download the result.

- **Application Logic Layer:**

This is the core logic of the app. It uses **Python** code and **Django Views** to handle the main operations. It includes:

- **Encryption/Decryption Module:** Supports both symmetric (e.g., AES) and asymmetric (e.g., RSA) encryption methods.

- **Hashing Module**: Computes secure hashes (e.g., SHA-256) of uploaded files.

- **Key Management Module:** Handles key generation, storage, retrieval, and security policies.

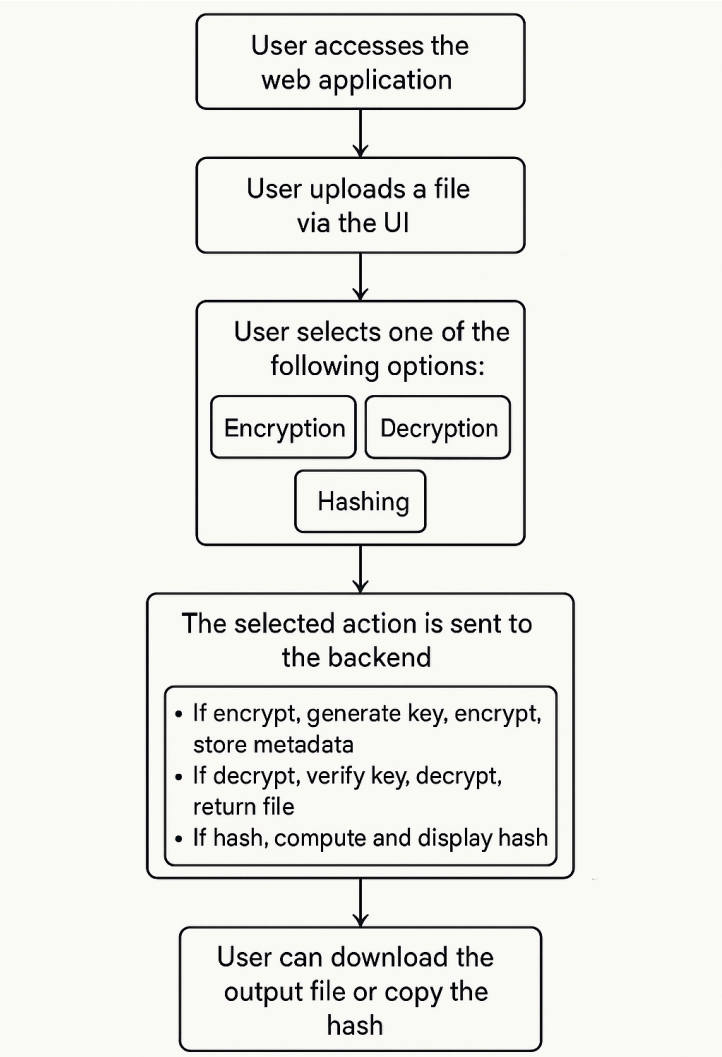
- **Data Layer:**

This layer uses **SQLite** or **PostgreSQL** to store metadata and the file system to temporarily save the files before downloading.

⇒ This architecture separates tasks clearly and makes it easier to fix issues or improve features in the future.

2.2 Design Diagram

The following diagram shows how the system's components work together, from user input to the final file output.



3. System Components & Messages/Data Exchanged

This section details what data each module takes, processes, and outputs, and how these modules communicate internally.

Component	**Input**	**Output**	**Messages Exchanged**
File Selection Module	File path from the user	Binary file data	Sends file data to the Operation Module
Operation Selector	Operation type from the user	None	Sends an operation request to Core Operations
Key Management Module	Key input from the user	Symmetric/ Asymmetric keys	Passes keys to encryption/ decryption/signing
Encryption Module	File data, encryption key	Encrypted file data	Receives file and key, sends encrypted data out
Decryption Module	Encrypted file, decryption key	Original file data	Receives file and key, sends plaintext out
Hashing Module	File data	Hash value	Sends the hash value to the output
Output/ Storage Module	Processed data (file/hash/ nature)	Saved file/displayed result	Receives the result, saves or displays it

4. Functional Flow

1. The user accesses the web application.
2. The user uploads a file via the UI.
3. User selects one of the following options: Encryption, Decryption, Hashing
4. The selected action is sent to the **backend**.
5. The backend processes the file:
    - If **encrypt**, generate key, encrypt, and store metadata.
    - If **decrypt**, verify key, decrypt, return file.
    - If **hash**, compute and display the hash.
6. The user can **download the output file** or **copy the hash.**

5. Roles (Actors)

- **Regular user:** can upload files, select operations (hash/encrypt/decrypt), and download the result.

6. Tools & Technologies

Tool/Library	Purpose
Python 3.x	Main programming language.
Django	A web framework to build the app.
HTML/CSS/JavaScript	Frontend templates and user interface
cryptography/hashlib	File encryption, decryption, and hashing.
SQLite (default)	Lightweight database for logs/history (optional).
Git/GitHub	Version control
VS Code	Development environment

7. Development Phases

Phase 1: Planning & Requirements
Define supported operations (AES, RSA, SHA-256), file handling flow, and tools needed.

Phase 2: Environment Setup
Initialize the Django project, configure templates, and install dependencies like PyCryptodome.

Phase 3: Backend Development
Implement hashing, encryption (symmetric & asymmetric), decryption, and key management modules.

Phase 4: Frontend Development
Create a simple UI for file upload, operation selection, and download using Django templates.

Phase 5: Integration & Testing
Connect the frontend and backend, test functionality, and ensure accuracy of cryptographic operations.

Phase 6: Security Enhancements
Add input validation, secure file handling, and basic error management.

Phase 7: Documentation
Prepare user and technical documentation, including setup and usage instructions.

Phase 8: Finalization
Polish the UI, clean the codebase, and prepare the final demo and deliverables.

8. Conclusion:

This document outlines the high-level design for a secure file management web application that supports encryption, decryption, and hashing. We present the system's architecture, workflows, user interactions, and the development tools selected.

Built on Django with AES and RSA encryption and SHA-256 hashing, the application ensures data security while offering an intuitive, educational user experience. Its modular structure defines clear interactions between components and follows a phased development plan.

The project bridges cryptography concepts with practical implementation, with the next phase focusing on building and testing the system to deliver a reliable, secure solution.