Secure Transfer: Design Document

1. Design Overview

SecureTransfer is a **web-based** file encryption, decryption, and hashing platform that enables both **full-file** and **partial encryption**. It simplifies secure file sharing through an intuitive user interface and powerful backend logic. Users can upload **files**, including **images**, for processing.

---

2. System Components and Exchanged Data

A. Frontend (Client-side)

Framework: React

Main Components:

- File Upload Interface
- Action Selector (Encrypt / Decrypt / Hash)
- Partial Encryption Viewer (text highlighter)
- Algorithm & Key Configuration Panel
- Progress & Notification Banners
- Result Preview and Download Page

Data Exchanged:

- File metadata
- File content (securely uploaded)
- User-selected encryption sections
- User-selected algorithm and key
- Final processed file or hash

B. Backend (Server-side)

Framework: Flask or Node.js

Main Components:

- File Parser & Analyzer
- Cryptographic Engine (AES, RSA, SHA-256, etc.)
- Partial Encryption Processor
- Key Generation & Validation Module
- File Integrity Checker
- API for frontend to communicate securely

Data Exchanged:

- Uploaded file data
- Section boundaries for partial encryption
- Keys (public/private/symmetric)
- Output files (encrypted/decrypted/hashed)
- Operation status updates

Encryption Models

- **Symmetric Encryption**
  - Classic: Caesar Cipher, XOR Cipher
  - Modern: DES, Triple DES, AES
- **Asymmetric Encryption**
  - RSA: Used for key exchange in hybrid encryption
- **Hybrid Encryption**
  - Combines RSA with AES or Triple DES for secure data encryption and key exchange

Decryption Tools

- Caesar and XOR Decryption
- DES and Triple DES Decryption
- AES Decryption (Fernet)

- **Hybrid Decryption**: Decrypt RSA-encrypted key, then decrypt data using AES or 3DES

---

3. User Roles, Functions, and Workflows

User Roles:

- **General User** (public): Can encrypt, decrypt, or hash files.
- **Admin/Dev Team** (internal): Manages cryptographic settings, logs, and user issues during development.

Workflows and Functions:

A. File Encryption (Full or Partial)

1. User uploads a file.
2. Selects "Encrypt" → Chooses algorithm (AES/RSA) → Sets or generates a key.
3. If "Partial Encryption":
   - Text file content preview is shown.
   - User selects sections to encrypt.
4. User confirms → File is sent to backend → Processed.
5. Processed file and summary are returned.
6. User downloads the secured file.

B. File Decryption

1. User uploads an encrypted file.
2. Selects "Decrypt" → Enters/decrypts key.
3. File sent to backend → Decryption performed.

4. Decrypted file returned with download option.

C. Hashing a File

1. User uploads file.
2. Chooses "Hash" → Selects algorithm (SHA-256 or BLAKE3).
3. Hash is computed and displayed.
4. User can copy or download the hash output.

---

4. Development Phases

Phase 1: Planning and Design

- Define modules, frontend layout, and encryption options

Phase 2: Core Development

- Implement classic, modern, asymmetric, and hybrid encryption logic
- Build React interface and backend API

Phase 3: Testing

- Encrypt/decrypt text, image, and document files
- Validate correctness of decryption and hashing
- Test partial encryption flow and hybrid system

Phase 4: Documentation

- User manual for system features
- Developer guide with inline comments

Phase 5: Deployment

- Deploy frontend
- Package backend with Flask and deploy as API

5. Tools and Technologies

Frontend

- **React.js** : Web interface for user interaction

Backend

- **Python**: Core logic implementation

Libraries

- `cryptography` (Fernet, AES)
- `hashlib` (SHA algorithms)
- `pycryptodome` (RSA, DES, Triple DES)
- Flask (lightweight web server and API handler)