Software Documentation

This folder consolidates architecture and component design diagrams and a concise setup guide for the SecureDove project.

Architectural Design (Summary)

- Patterns: Client–Server, Layered architecture. Server uses modular routing (controller-like routes), middleware for cross-cutting concerns (auth, rate limiting), and a persistence layer (SQLite). Client uses React Context/Provider with hooks for state and side effects; APIs wrap REST; WebSocketContext provides realtime.

- Trust model: Zero-knowledge server for message content and private keys. All message payload crypto is client-side (AES-GCM), keys wrapped per participant (RSA-OAEP).

Component-Level Design (Summary)

- Client components expose clear interfaces via Contexts for UI consumption; APIs encapsulate HTTP; WebSocketContext encapsulates Socket.IO. Utilities (Crypto/Storage/Backup) are pure helpers.

- Server endpoints are grouped by concern (auth, contacts, conversations, messages). Middleware enforces auth and throttling. Realtime gateway emits per-conversation events.

Installation

Prerequisites

- Node.js LTS (v18+ recommended)

- npm (bundled with Node)

Clone and install

- `npm install` in both `server/` and `client/` directories.

Server environment

- Create `server/.env` with (example defaults):

  - `PORT=8000`

  - `NODE_ENV=development`

  - `JWT_SECRET=<set-a-strong-secret>`

- `DB_PATH=./database/securedove.db`

- `CORS_ORIGIN=http://localhost:5173`

- `RATE_LIMIT_WINDOW_MS=900000` and `RATE_LIMIT_MAX_REQUESTS=100`

- `LOGIN_RATE_LIMIT_WINDOW_MS=900000` and `LOGIN_RATE_LIMIT_MAX_REQUESTS=5`

Client environment

- Create `client/.env` with:

- `VITE_API_URL=http://localhost:8000/api`

- Optionally: `VITE_SOCKET_URL=http://localhost:8000`

Running

Option A: Start both via helper script

- From repo root: `./start.sh` (or `start.bat` on Windows)

Option B: Start separately

- Server: `cd server && npm run dev` (or `npm start`)

- Client: `cd client && npm run dev` (Vite dev server on port 5173)

Usage

1) **Register and Login**

- Registration generates an RSA keypair client-side; the private key is encrypted with a password-derived key and stored server-side only in encrypted form.

- Login decrypts the private key client-side after JWT authentication.

2) **Contacts**

- Add/remove/list contacts; fetch public keys for secure key wrapping.

3) **Conversations**

- Create conversations by wrapping a content key per participant. Add participants either by:

- Sharing history (re-wrap historical content keys for new users), or

- Rotating to a new content key (key number increments for all participants).

- Leave/delete removes the current user’s membership; emits a system event.

4) **Messaging**

- Messages are encrypted (AES-GCM) client-side with the conversation content key. Realtime delivery via Socket.IO; history is fetched via REST and decrypted locally.

- Edit/delete operations update or remove encrypted payloads; system events (participant added/removed, key rotation) appear as broadcast messages in the timeline.

5) **Backup & Local Data**

- Create/export a backup (JSON) of encrypted messages and metadata. Import/restore to merge/replace local data. Optionally clear all local messages.

Notes

Security

- The server never handles plaintext messages or private keys.

- Ensure HTTPS and strong JWT secret in production.

CORS/WebSocket

- Match `CORS_ORIGIN` to the client dev server (`http://localhost:5173`) and `VITE_SOCKET_URL` to the server origin.