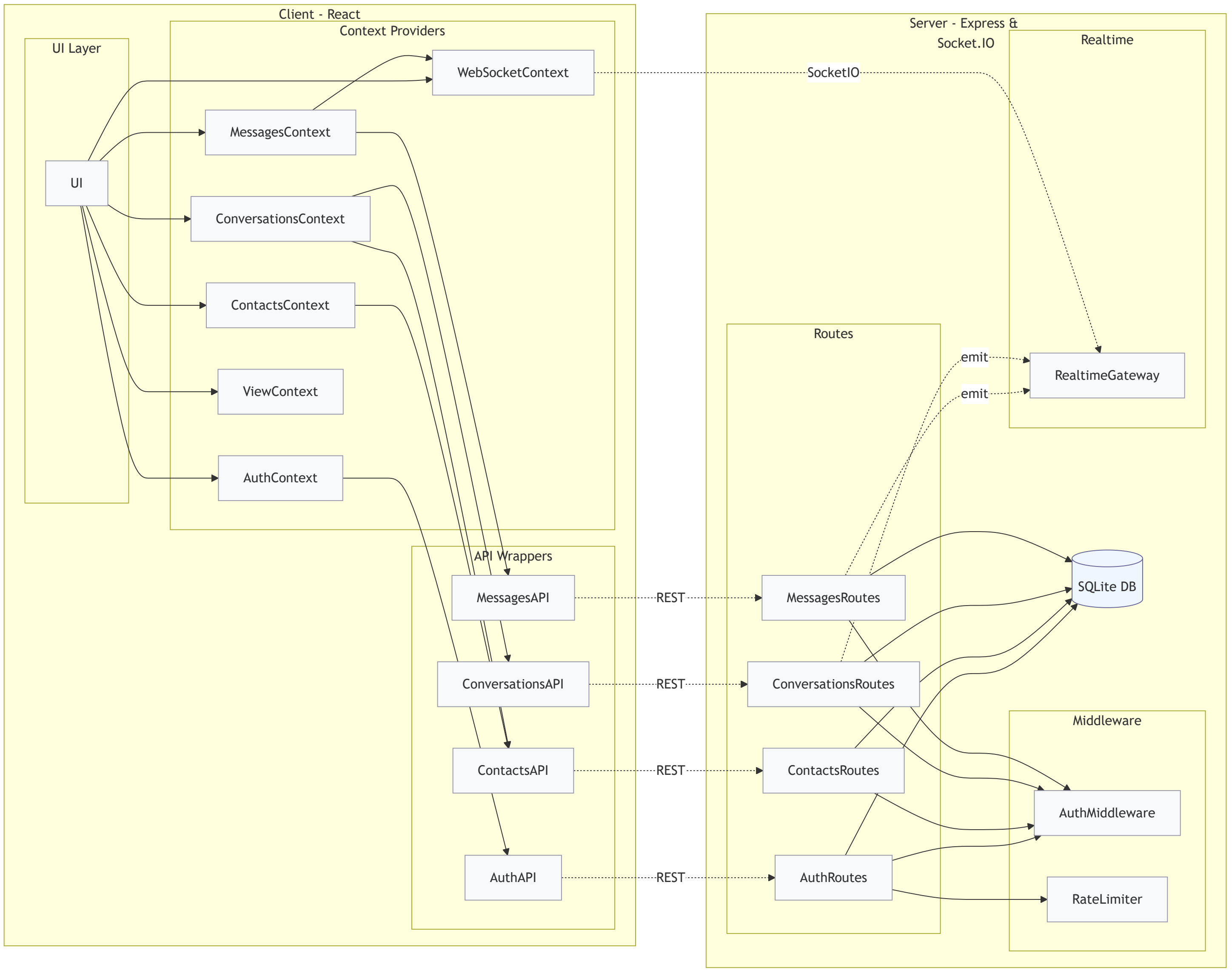
SecureDove Architecture

This document summarizes the architecture and components using only information from this `documentation/` folder. It includes Mermaid versions of the PlantUML diagrams and concise supporting text based on `documentation/README.md`.

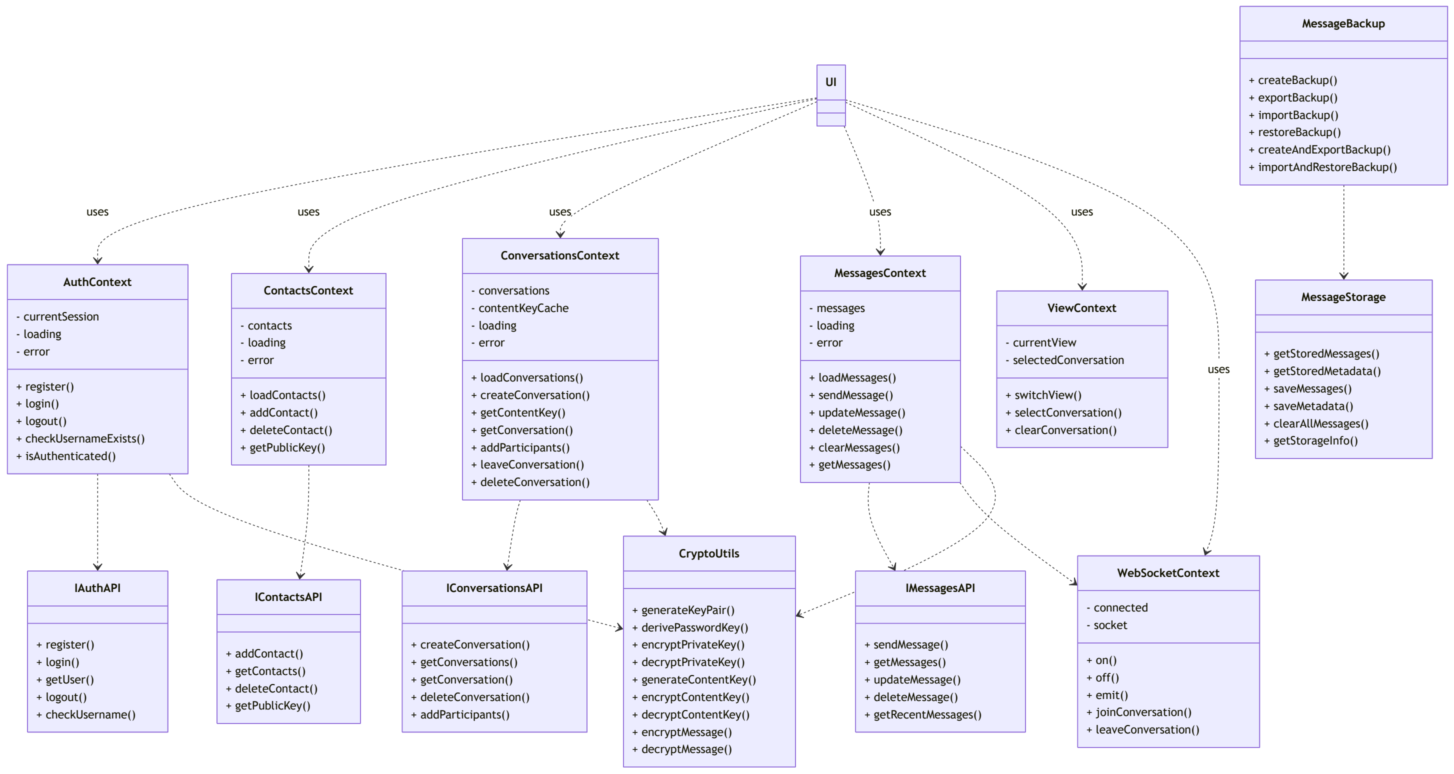
Overview

- Patterns: Client–Server, Layered. The server uses modular routing (controller-like routes) with middleware for cross-cutting concerns and a persistence layer. The client uses React Context/Provider with hooks; REST API wrappers and a WebSocket provider support data and realtime.
- Trust model: Zero-knowledge server for message content and private keys. All cryptography for message payloads is client-side.

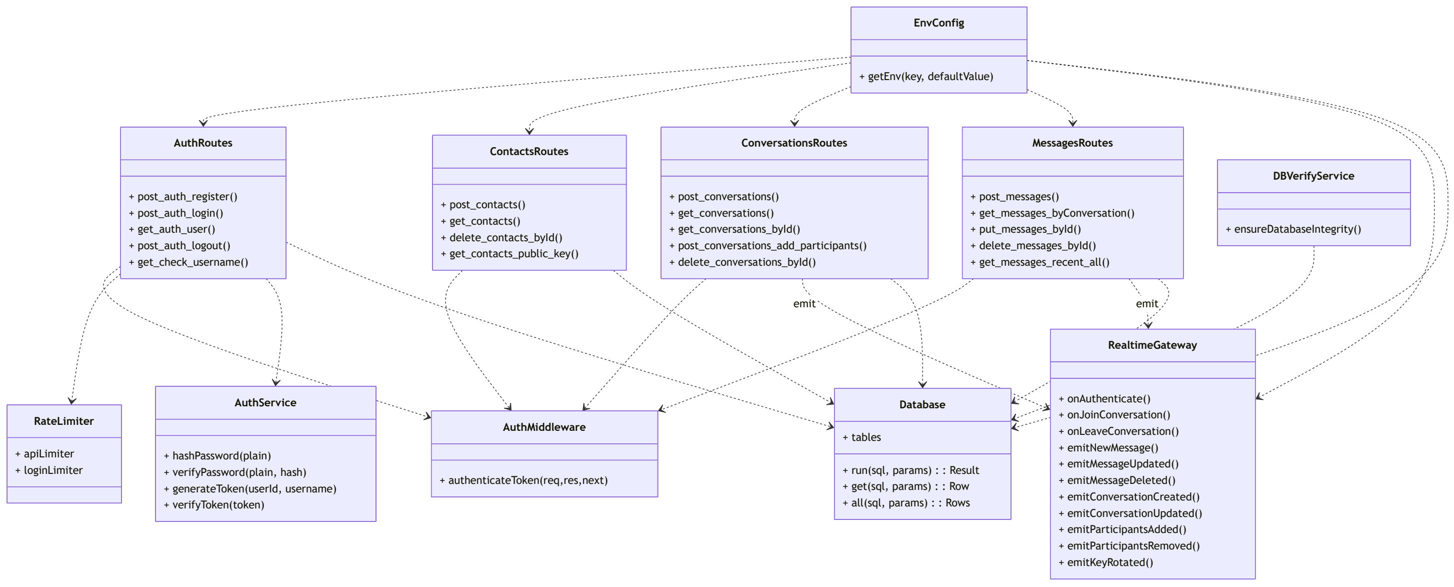
Architecture Brief



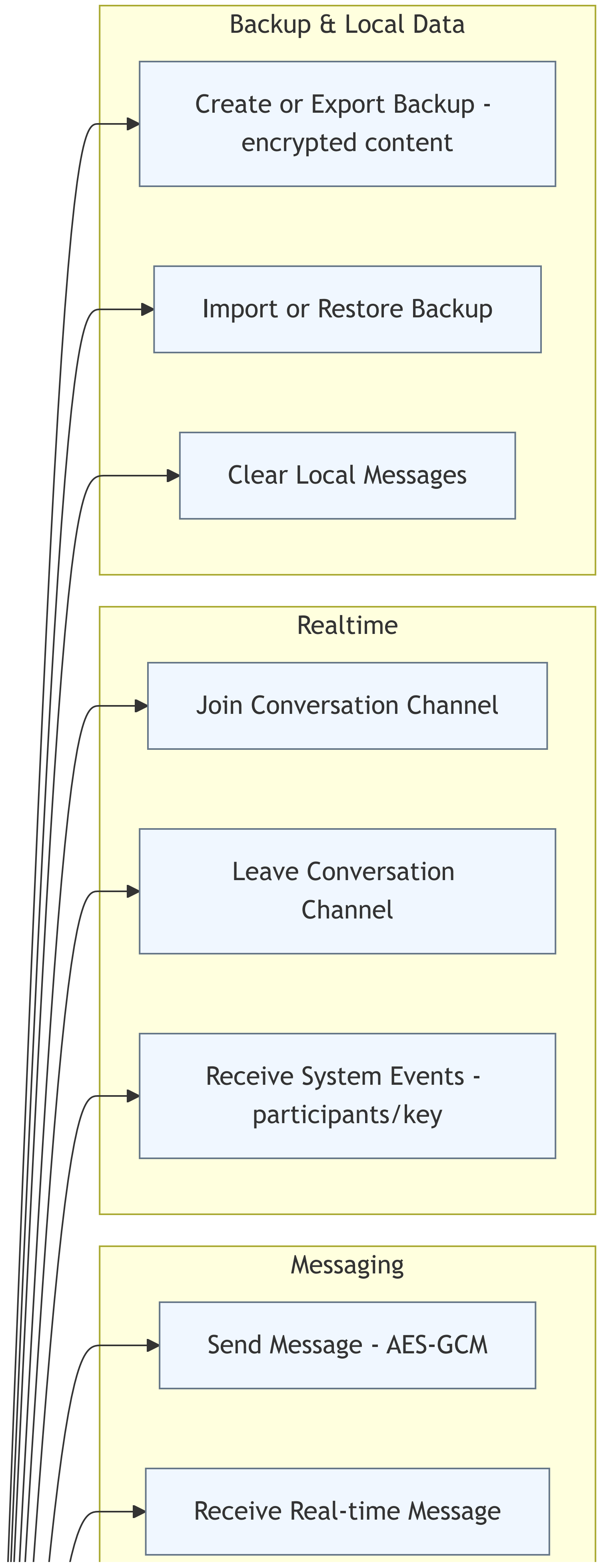
Client Components (Elaborated)

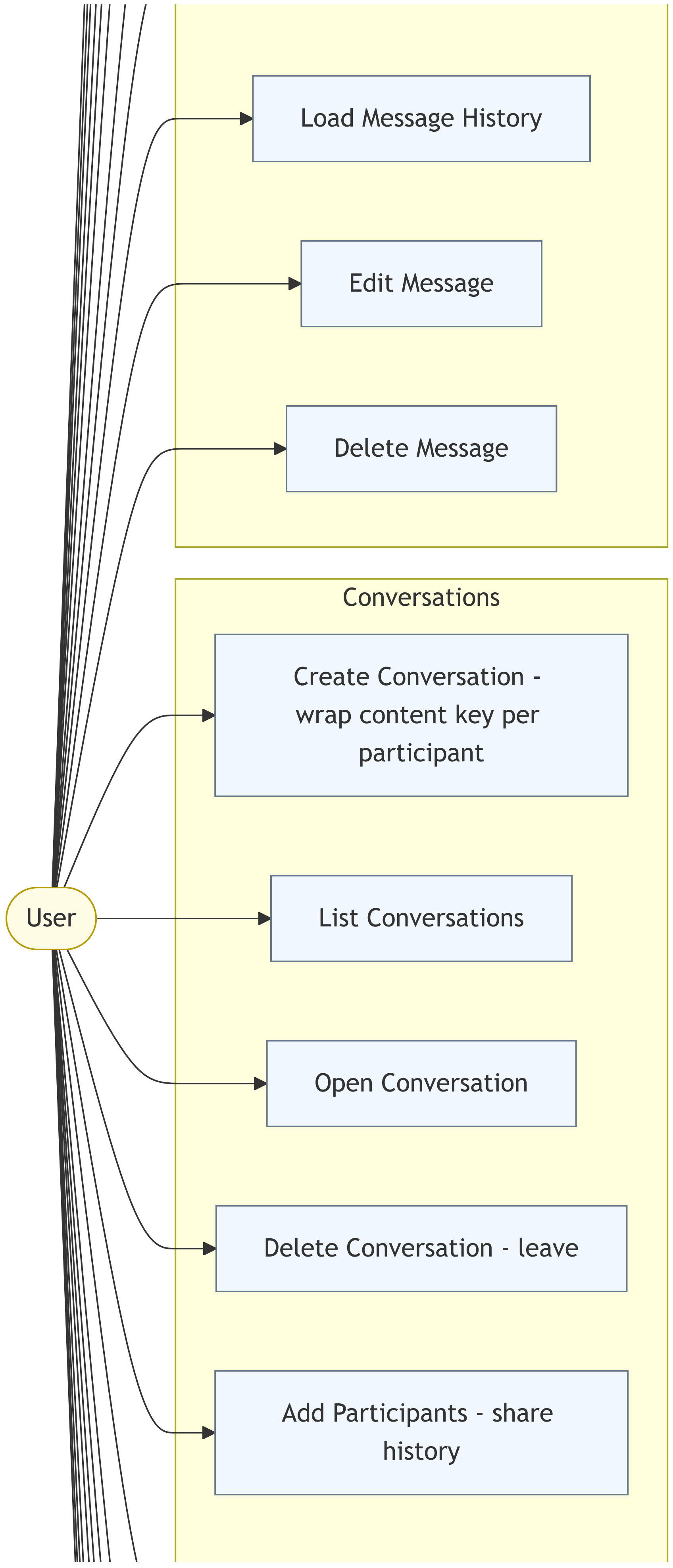


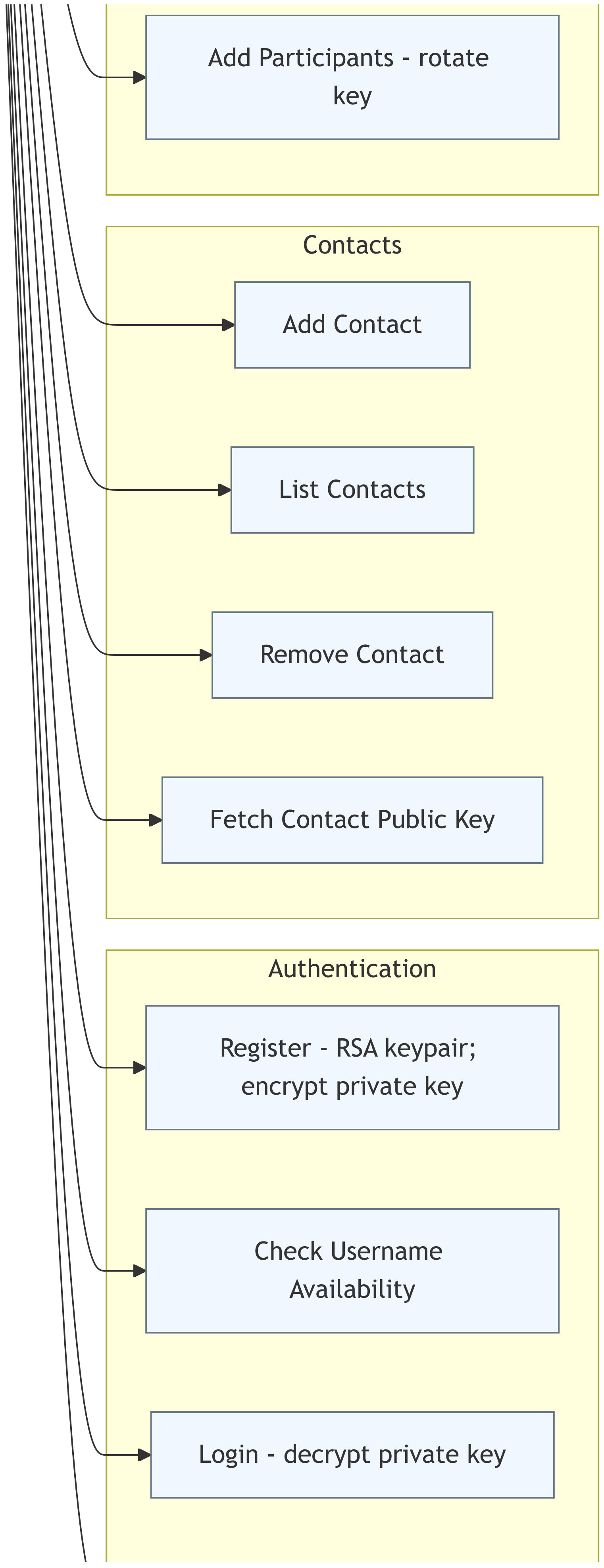
Server Components (Elaborated)

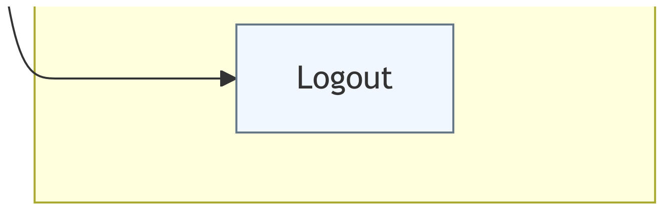


User Use Cases


Backup & Local Data
Create or Export Backup - encrypted content
Import or Restore Backup
Clear Local Messages
Realtime
Join Conversation Channel
Leave Conversation Channel
Receive System Events - participants/key
Messaging
Send Message - AES-GCM
Receive Real-time Message


Load Message History
Edit Message
Delete Message
Conversations
Create Conversation - wrap content key per participant
User
List Conversations
Open Conversation
Delete Conversation - leave
Add Participants - share history


Add Participants - rotate key
Contacts
Add Contact
List Contacts
Remove Contact
Fetch Contact Public Key
Authentication
Register - RSA keypair; encrypt private key
Check Username Availability
Login - decrypt private key



Installation

Prerequisites

- Node.js LTS (v18+ recommended)
- npm (bundled with Node)

Clone and install

- Run `npm install` in both `server/` and `client/` (see `documentation/README.md`).

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`

Database

- Initialize SQLite:
  - `cd server && npm run init-db`
  - Verify (optional): `npm run verify-db`

Running

Option A: helper script

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

Option B: separate

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

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 with per-participant wrapped content keys. Add participants either by:
  - Sharing history (re-wrap historical keys), or
  - Rotating to a new content key (incremented key number).
- Leave/delete removes current user membership and emits a system event.

4. Messaging

- Messages are encrypted client-side (AES-GCM) with the conversation content key. Realtime delivery uses Socket.IO; history is fetched via REST and decrypted locally. Edit/delete operations update or remove encrypted payloads; system events appear as broadcast items.

5. Backup & Local Data

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

Notes

- Security: The server never handles plaintext messages or private keys; use HTTPS and a strong JWT secret in production.
- CORS/WebSocket: Match `CORS_ORIGIN` to the client dev server and `VITE_SOCKET_URL` to the server origin.