GNOMAN

CHRISTOPHER HIRSCHAUER

Gnoman 2.0 User Manual

First edition

This book was professionally typeset on Reedsy.
Find out more at reedsy.com

Contents

1

Gnoman 2.0 Desktop Application

Gnoman is a cross-platform Electron desktop application that combines a local Express API with a React renderer to manage Gnosis Safe workflows from a single secured workspace. The project is written entirely in TypeScript and ships with tooling for simulating Safe transactions, managing wallets, and enforcing registration policies before operators can act on production Safes.

Tech stack

- **Electron 28** for the desktop shell, preload isolation, and IPC keyring bridge (main/).
- **Express** with TypeScript for the local API that powers wallet, Safe, sandbox, and registration flows (backend/).
- **React + Tailwind (Vite)** for the renderer UI (renderer/).
- **Better SQLite3** for persisting registration state and transaction holds under a local .safevault/ directory.
- **Ethers v6** for wallet creation, encryption, and contract simulation utilities.

Repository layout

```
/ (root) ├───
 backend/              # Express API, services, and
 route handlers ├───
 main/                 # Entrypoint, preload, and OS
 keyring ├───
 modules/sandbox/      # Shared sandbox engine, ABI
 parser,UI ├───
 renderer/             # React renderer bundled with
 Vite ├───
 scripts/              # Build utilities for
 packaging renderer ├───
 docs/                 # Markdown documentation for
 app & wiki ├───
 tests/                # API smoke tests and
 fixtures ├───
```

```
package.json          # Root npm scripts and
dependencies └───────
tsconfig*.json        # TypeScript project references
```

Prerequisites

RequirementNotes

Node.js 18+

Tested with the LTS release bundled with npm 9

npm 9+

Installed with Node.js

SQLite

Provided by better-sqlite3; native build tools (Xcode Command Line Tools / build-essential / Windows Build Tools) may be required on first install

Local fork tool (optional)

anvil or another Hardhat-compatible command for sandbox forking

Installation

```
npm install
(cd renderer && npm install)
```

Development workflow

The backend listens on http://localhost:4399 by default. Run the services in separate terminals:

```
npm run dev:backend    # Start the Express API with
ts-node-dev
npm run dev:renderer   # Launch the Vite dev server
for the renderer UI
```

You can also run both web stacks together:

```
npm run dev            # concurrently runs
dev:backend and dev:renderer
```

To open the Electron shell, build the TypeScript bundles and launch the desktop window:

```
npm run dev:electron   # Builds backend/main/renderer
then boots Electron
```

Production build

```
npm run build          # Compile backend, main
process, and renderer
npm start              # Launch Electron with the
bundled renderer
```

Additional scripts

Script	**Description**
npm run clean	Remove the dist/ directory
npm run lint	Run ESLint across backend, main, renderer, and modules
npm run build:backend	Compile the Express API to dist/backend
npm run build:main	Compile the Electron main process to dist/main
npm run build:renderer	Build the renderer UI (renderer/dist)

Backend API summary

All endpoints are served from http://localhost:4399/api.

Health

- GET /health – service heartbeat with current timestamp.

Wallets (backend/routes/walletRoutes.ts)

- GET /wallets – list stored wallet metadata.
- POST /wallets/generate – create a new encrypted wallet.
- POST /wallets/import/mnemonic – import a wallet from a mnemonic phrase.
- POST /wallets/import/private-key – import a wallet from a raw private key.
- POST /wallets/vanity – brute-force vanity address genera-

tion with prefix/suffix filters.
- POST /wallets/:address/export – decrypt and export an encrypted JSON keystore for a stored wallet.

Wallet metadata and encrypted secrets live in-memory for now. Exports are re-encrypted with ethers.Wallet.encrypt so that secrets never leave the API unprotected. The UI currently surfaces wallet creation and listing from the /wallets page.

Safes (backend/routes/safeRoutes.ts)

- POST /safes/load – connect to a Safe on a specified RPC URL.
- GET /safes/:address/owners – list cached Safe owners.
- POST /safes/:address/owners – add an owner and update the threshold.
- DELETE /safes/:address/owners/:ownerAddress – remove an owner and update the threshold.
- POST /safes/:address/threshold – change the approval threshold.
- POST /safes/:address/modules – enable a Safe module.
- DELETE /safes/:address/modules/:moduleAddress – disable a module.
- POST /safes/:address/transactions – register a transaction proposal and enforce hold policy tracking.
- POST /safes/:address/transactions/:txHash/execute – execute a stored transaction (respecting hold timers).
- POST /safes/:address/hold/toggle – enable or disable the hold policy for a Safe.
- GET /safes/:address/transactions/held – list transactions currently under the hold policy.

Transactions and Safe metadata are kept in-memory while hold-state metadata is persisted to SQLite under .safevault/holds.sqlite.

Sandbox (backend/routes/sandboxRoutes.ts)

- POST /sandbox/call-static – legacy helper for single call-Static simulations using ad-hoc ABI JSON.
- POST /sandbox/contract/abi – parse and cache contract ABI definitions.
- GET /sandbox/contract/abis – list cached ABIs.
- POST /sandbox/contract/simulate – run contract simulations with decoded return data, gas estimates, and traces.
- POST /sandbox/contract/safe – execute Safe-specific simulations with the canonical Safe ABI.
- GET /sandbox/contract/history – retrieve the most recent simulation results for replay.
- DELETE /sandbox/contract/history – clear the persisted simulation history.
- POST /sandbox/fork/start – spawn a local fork (defaults to anvil) for simulations.
- POST /sandbox/fork/stop – stop the active fork.
- GET /sandbox/fork/status – inspect fork process status.

The sandbox writes JSON logs to modules/sandbox/logs/ and coordinates optional local fork lifecycles.

Product registration (backend/routes/registrationRoutes.ts)

- GET /registration – fetch current registration status.
- POST /registration – store a hashed product license and registration email using scrypt hardening.

Registration data persists in .safevault/registration.sqlite and must match the previously registered license to update the email address.

Desktop application features

- **Dashboard** – high-level overview of stored wallets and the currently connected Safe.
- **Wallets** – generate encrypted wallets with optional aliases, hidden flag, and password overrides, then list stored metadata.
- **Safes** – connect to a Safe, review owners/modules, and monitor transactions held under the enforced delay window.
- **Sandbox** – switch between the legacy Safe callStatic form and the advanced sandbox panel powered by modules/sandbox/ui. Upload or paste ABIs, choose functions, provide parameters, replay historical simulations, and manage an optional local fork.
- **Keyring** – interact with the Electron IPC bridge (window.safevault) to list and reveal secrets stored in the OS keyring (with an in-memory fallback when keytar is unavailable).
- **Settings** – register the product license, view current registration status, and jump to the in-app wiki.
- **Wiki Guide** – render Markdown documentation from doc-

s/wiki directly inside the renderer.

Data directories & security

- Registration and transaction-hold records are stored under .safevault/ in the project working directory.
- Sandbox logs persist to modules/sandbox/logs/ for replay and auditing purposes.
- Wallet private keys stay encrypted in-memory using AES-256-GCM with PBKDF2 key derivation. Exported keystores require the caller-supplied password.
- The Electron preload exposes a minimal window.safevault.invoke surface to keep privileged operations isolated from the renderer context.

Documentation

Additional guides live under docs/. Start with docs/user-guide.md for a comprehensive walkthrough and docs/wiki/ for content surfaced in the in-app knowledge base.

2

Gnoman 2.0 Wiki User Guide

Welcome to the Gnoman user guide. This wiki outlines the workflows that keep Safe owners in control of their assets while maintaining strong operational security.

Getting Started

1. **Install dependencies** using npm install at the repository root and within renderer/.
2. **Start the backend** with npm run dev:backend to unlock wallet and Safe orchestration APIs.
3. **Launch the renderer** with npm run dev:renderer or boot the Electron shell via npm run dev:electron.
4. **Register your product license** inside the application under **Settings → Product Registration** to enforce organization-wide compliance policies.

Wallet Management

- Generate, import, and export wallets from the **Wallets** tab. Secrets are encrypted with AES-256-GCM before touching disk.
- Use the vanity search tools to derive predictable addresses without revealing mnemonic phrases.
- Store aliases and metadata securely through the **Keyring** page, which leverages the host operating system keyring when available.

Safe Operations

- Review Safe owners, thresholds, and modules inside the **Safes** dashboard.
- Toggle the 24-hour transaction hold policy to force human-in-the-loop approvals for high-impact operations.
- Exercise the **Sandbox** for deterministic callStatic simulations before broadcasting any transaction bundle.

Security Checklist

- Validate workstation compliance by confirming product registration upon first launch.
- Rotate RPC credentials regularly and audit all connected services.
- Keep application dependencies patched and monitor release notes for security advisories.
- Protect the .safevault/ directory with OS-level full disk encryption.

Need More Help?

Enhance this wiki with organization-specific procedures by adding Markdown files to docs/wiki/. Share PRs with your operations team to keep everyone aligned on Gnoman best practices.

3

Gnoman Wiki User Guide

This in-app wiki distills the core workflows and best practices for operating Gnoman securely. Use it as a quick reference while the desktop application is running.

Getting started

1. **Start the backend** with npm run dev:backend. The renderer expects the API on http://localhost:4399.
2. **Launch the renderer** with npm run dev:renderer or boot the full Electron shell via npm run dev:electron to access the keyring bridge.
3. **Register the product** inside **Settings → Product Registration** to store a scrypt-hardened license and email address under .safevault/registration.sqlite.

Wallet management

- Generate wallets from the **Wallets** tab. Each request calls the backend to create a new key pair, encrypt the private key with AES-256-GCM, and return metadata.
- Record the generated password (or provide your own) so you can export the wallet later via the API.
- Hidden wallets are marked for keyring storage; when keytar is available, secrets are written to the OS keychain instead of disk.

Safe operations

- Connect to an existing Safe from the **Safes** tab by supplying the Safe address and RPC URL. The backend verifies the network before caching owners, modules, and threshold.
- Monitor the **Held Transactions** panel to see proposals subject to the enforced hold period. Hold timers persist in .safevault/holds.sqlite.
- Use the backend endpoints to add/remove owners, change thresholds, and manage modules as required by your operational policies.

Sandbox simulations

- The **Sandbox** tab hosts two tools:
- A quick Safe callStatic form for validating guard contracts or Safe modules.
- An advanced panel (from modules/sandbox/ui) that lets you load ABIs, choose functions, provide parameters, replay previous simulations, and run against a local fork (defaults

to the anvil command).
- Simulation results are saved as JSON in modules/sandbox/logs/ so you can audit or replay them later.

Keyring & secrets

- The **Keyring** view lists stored aliases via the Electron preload bridge (window.safevault).
- Select **Reveal** to fetch a secret securely from the OS keyring. In development environments without keytar, SafeVault transparently falls back to an in-memory store so testing can continue.

Security checklist

- Ensure product registration succeeds before managing production Safes.
- Rotate RPC credentials regularly and validate that your fork command (e.g., anvil) is patched.
- Keep dependencies updated and review release notes for security advisories.
- Protect the .safevault/ directory with OS-level full-disk encryption.

Stay aligned with your organization’s procedures by extending this wiki with additional Markdown files under docs/wiki/.

4

PROPRIETARY LICENSE AGREEMENT — GNOMAN 2.0 SECURITY & ADMINISTRATION PLATFORM

Version: 1.0 Effective Date: 10.26.2025

Licensor: C. Hirschauer, an individual residing in Iowa ("Licensor")

Licensee: The purchaser, recipient, or authorized operator of a valid GNOMAN 2.0 license ("Licensee").

1. Definitions

1.1 **"Software"** means the GNOMAN 2.0 desktop application, associated executables, libraries, Electron/React renderer, backend services, modules, APIs, plug-ins, data stores, simulation tools, key-management utilities, documentation, and any updates, patches, or derivatives provided by Licensor (including object and source code).

1.2 **"Documentation"** means all user guides, technical manuals, wiki pages, and other written or electronic materials accompanying the Software.

1.3 **“Personal License”** means a non-transferable license limited to one natural person for personal or educational use only.

1.4 **“Business License”** means a non-exclusive license to deploy and operate the Software for the internal operations of one legal entity (including its employees and contractors under NDA), subject to the limits of this Agreement.

1.5 **“License Fee”** means the non-refundable payment required to obtain a License.

1. 6 **“License Token”** means a digitally-signed activation string generated by Licensor’s Ed25519 keypair and verified by the Software offline through its embedded public key.

2. Grant of License — Limited and Conditional

2.1 **Personal License.** Upon full payment and acceptance of this Agreement, Licensor grants Licensee a limited, non-transferable, non-sublicensable, revocable license to install and use one copy of the Software solely for personal, non-commercial purposes on a single machine.

2.2 **Business License.** Upon full payment and compliance, Licensor grants Licensee a limited, non-transferable (except as provided in §16), non-sublicensable, revocable license to use the Software solely for internal business operations on the number of devices or environments purchased.

2.3 **Reservation of Rights.** All rights not expressly granted remain with Licensor. The Software and Documentation are licensed, not sold.

3. Prohibited Uses — Zero Tolerance

Licensee shall not, and shall not permit any person to:

a) Copy, distribute, rent, lease, sell, sublicense, or transfer the Software or Documentation;

b) Reverse-engineer, decompile, disassemble, or modify the Software (except as required by law after notice to Licensor);

c) Use the Software to offer third-party services (e.g., hosting, SaaS, multi-tenant management) without a separate commercial agreement;

d) Remove or obscure any copyright or proprietary markings;

e) Bypass, disable, or tamper with license-verification or cryptographic controls, including Ed25519 signature checks or token validation;

f) Integrate the Software with open-source components in a manner that would require disclosure of its source code; or

g) Use the Software in violation of any law or third-party rights.

4. Delivery, Activation, and Fees

4.1 **Delivery.** The Software is delivered electronically after receipt of payment and identity verification (if applicable).

4.2 **Fees.** License Fees are non-refundable unless expressly agreed in writing. Failure to pay when due constitutes material breach.

4.3 **Activation.** Each license requires a valid offline License Token digitally signed by Licensor. The Software verifies the token using its embedded public key without internet communication. Tampering with this mechanism voids the License.

5. Ownership and Intellectual Property

Licensor retains all title, ownership, and intellectual-property rights in the Software and Documentation. This Agreement conveys only the limited rights expressly granted.

6. Audit and Verification

Licensor may request reasonable evidence of proper license activation or valid License Token(s). Licensee must cooperate by providing token files or logs on request. Non-compliance is a material breach.

7. Support and Updates

Support or maintenance is not included unless purchased separately. Licensor may release updates at its discretion. Nothing herein obligates continued development or compatibility with future platforms.

8. Confidentiality

Licensee shall treat the Software, Documentation, and license utilities as confidential, protecting them from unauthorized disclosure with no less care than its own confidential information.

9. Warranties and Disclaimers

9.1 **Limited Warranty.** For thirty (30) days after delivery, Licensor warrants the Software will substantially conform to the Documentation. Remedy: repair or replacement at Licensor's option.

9.2 **Disclaimer.** EXCEPT AS SET FORTH ABOVE, THE SOFTWARE IS PROVIDED "AS IS." ALL OTHER WARRANTIES, EXPRESS OR IMPLIED (INCLUDING MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE), ARE DISCLAIMED TO THE MAXIMUM EXTENT ALLOWED BY LAW.

10. Indemnification

Licensee shall indemnify and hold Licensor harmless from all claims, damages, or expenses (including attorneys' fees) arising from Licensee's use or breach of this Agreement.

11. Limitation of Liability

TO THE MAXIMUM EXTENT PERMITTED BY LAW, LICENSOR WILL NOT BE LIABLE FOR INDIRECT, INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES (INCLUDING LOSS OF PROFITS OR DATA). TOTAL LIABILITY WILL NOT EXCEED THE LICENSE FEE PAID BY LICENSEE.

12. Remedies and Equitable Relief

12.1 **Injunctive Relief.** Breach of Sections 3, 5, or 8 causes irreparable harm. Licensor may seek injunction or specific performance without bond.

12.2 **Liquidated Damages.** Each intentional breach of Sections 3 or 8 incurs liquidated damages of USD $50,000 (or greater actual damages proven), plus costs and attorneys' fees.

13. Enforcement and Attorneys' Fees

The prevailing party in any action arising from this Agreement is entitled to reasonable attorneys' fees and costs, including appeals and collection.

14. Termination

Upon any material breach, this Agreement terminates immediately. Licensee must cease all use, destroy copies, and certify destruction. Licensor may remotely disable license verification or invalidate tokens where lawfully permitted.

15. Export and Compliance

Licensee shall comply with all export, import, and sanctions laws. The Software shall not be exported to prohibited regions or persons.

16. Transfer and Assignment

Licenses are non-transferable without written consent. Business License assignment to a successor in interest requires Licensor's prior approval and assumption of this Agreement.

17. Severability

If any provision is held invalid, the remainder remains in force. The parties shall replace invalid terms with ones closest in intent and effect.

18. Governing Law and Venue

This Agreement is governed by the laws of [GOVERNING JURISDICTION]. Exclusive venue for disputes shall be the state or federal courts of [SELECTED VENUE]. Each party waives objection to inconvenient forum.

19. Notices

All notices must be in writing and sent to the addresses specified in the order or invoice. Notices take effect upon receipt.

20. Entire Agreement and Amendment

This document is the entire agreement regarding the Software and supersedes all prior understandings. Amendments require a written instrument signed by both parties.

21. Miscellaneous

21.1 **No Implied Licenses.** None are granted by implication or estoppel.

21.2 **Publicity.** Licensee shall not use Licensor's name or trademarks without written consent.

21.3 **Survival.** Ownership, confidentiality, indemnity, liability, and other provisions that by their nature should survive termination will do so.

22. Signature and Acceptance

By installing, activating, or using GNOMAN 2.0, Licensee acknowledges having read and agreed to be bound by this Agreement.

Activation of the Software using a valid Ed25519-signed License Token constitutes electronic acceptance and execution of this Agreement.

(End of Agreement — GNOMAN 2.0 v1.0)