# conceal er

**THE AI SECRET MANAGER**

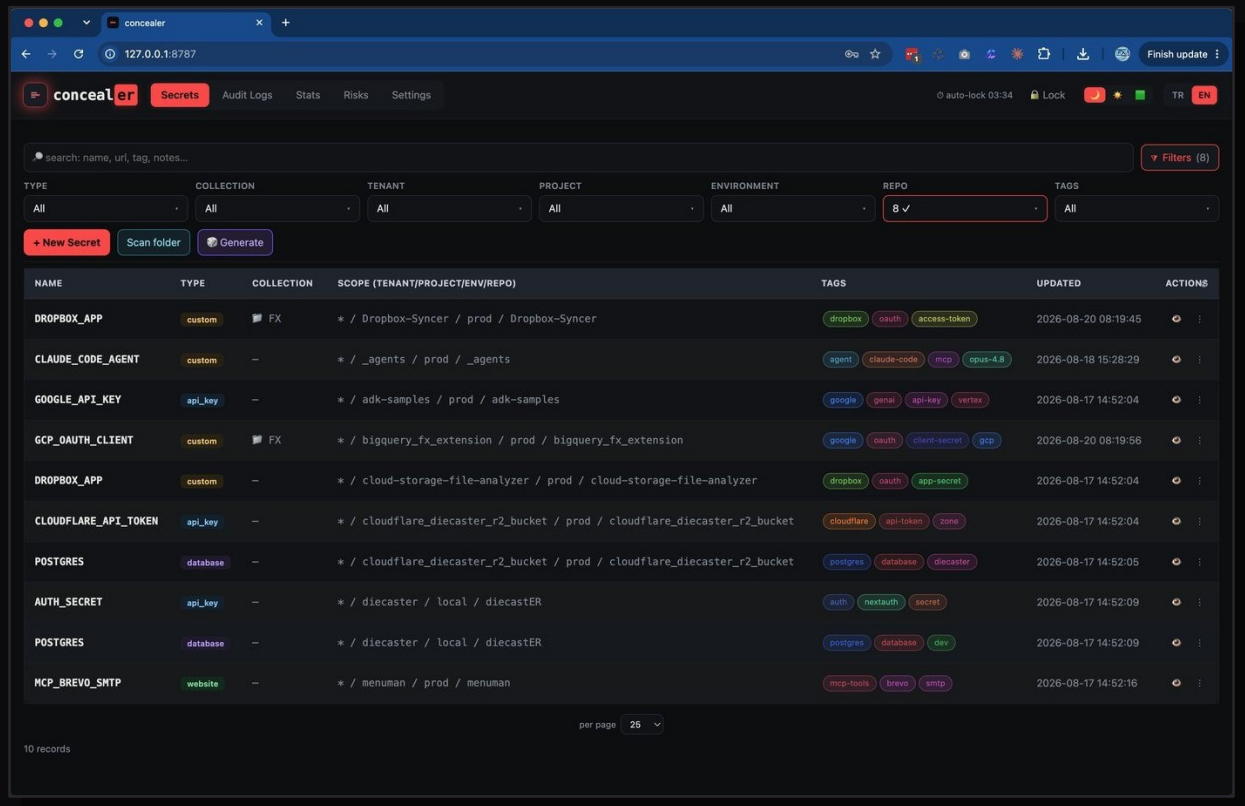
## Stop pasting keys into chat windows.

Your agents need secrets. Handing them the real key is how it leaks. concealer keeps one encrypted file on your own disk — SOPS and age underneath — and gives every interface, including an MCP server your AI can call, a way to use a secret without ever seeing it.

CLI · WEB UI · TUI · MCP SERVER · CHROME

concealer.fxerkan.com

MIT · macOS · Linux · Windows




Every secret, scoped and tagged — and none of it leaves this machine

# And it tells you what already leaked.

A secret you rotate is safe. A secret sitting in last year's commit is not. concealer scans the working tree, the git history, .env files and imports, and lists what is already public as exposure — not as a warning you can dismiss.

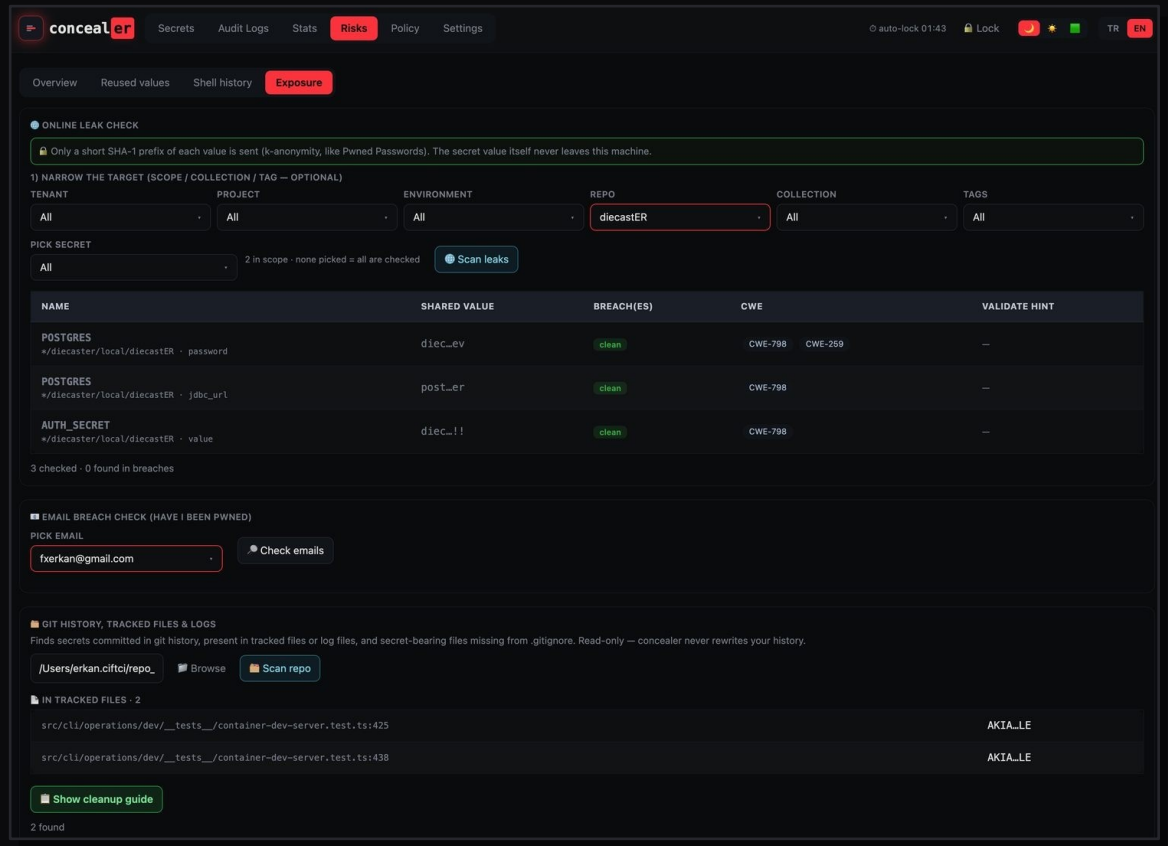
```
$ cer scan ./repo --history --envvars --import
```

```
$ cer expose
```

```
$ cer gitscan
```

The same scan runs from the web UI, and every rotation is written to a tamper-evident audit chain.

Source: docs/security.md




Risks → Exposure

# Your coding assistant reads every file in the repo.

Claude Code, Codex, Gemini CLI, opencode, Cursor — all of them are wonderful, and all of them read your project files. The moment a key lands in a .env, a credentials.json, or a chat window, it has four ways out.

1. **Read by the agent** and sent to a model provider.
2. **Captured in logs or telemetry** of whatever tool you happen to be using.
3. **Committed to git** by accident — and it stays in the history.
4. **Copied across a dozen repos** under the same name, with no way to tell them apart.

## What that costs you

| 4 | 1 | 0 | ∞ |
|---|---|---|---|
| WAYS A KEY ESCAPES | PASTE YOU'LL REGRET | TOOLS THAT STOP IT IN THE TRANSCRIPT | REACH ONCE IT'S OUT |

- Nothing on this list is a bug in the agent — it is what reading your files *means*.
- So the fix cannot be "be careful". It has to be that the value is never in the file, the transcript, or the history in the first place.
- If you have ever pasted a secret into a chat window and regretted it a second later — that is the itch concealer scratches.

Source: README.md

# Doppler, Infisical, Vault and 1Password all solve a different problem.

They are built for fleets, teams and sync. A single developer with an AI agent on one laptop needs five things none of them lead with:

01 · Offline

### Local-only, provably

Secrets never leave the machine. No account, no sync server, no telemetry — you can put a firewall in front of it and watch nothing phone home.

02 · Portable

### Password, not hardware

The vault decrypts on any machine with just the master password. It is not tied to this laptop's Keychain or TPM. Copy the files, type the password, done.

03 · Git

### Inspectable, versionable

The encrypted vault is a plain SOPS file you can commit — keys visible, values encrypted. The tool itself is one readable Python script, not a black box.

04 · AI-safe

### Agents use, never see

An MCP server lets an agent list names and inject values into a command's environment. Plaintext is redacted from everything it reads back.

05 · Scoped

### One place, many projects

The same value across many repos is disambiguated by tenant / project / environment / repo instead of one ambiguous name.

06 · Honest

### Documented ceilings

Every limit is written down, not hidden: the local audit key, CPython's un-zeroed memory, the single-user web server. A security review can read them.

Source: README.md

# A thin, auditable wrapper around two battle-tested tools.

concealer implements no cryptography of its own. SOPS encrypts the vault; age is the backend. concealer adds only the part that was missing: typed secrets, four-dimension scoping, tags, a professional local web console, a tamper-evident audit chain, and an MCP server.

- **One encrypted file.** `secrets.enc.yaml` — per-value AES-256-GCM over age X25519, git-friendly.
- **One readable script.** Python 3, standard library only. MIT licensed. No `pip install` for the tool itself.
- **Five interfaces, one core.** CLI · Web UI · TUI · MCP · Chrome extension all funnel into the same code path — and the same audit log.
- **Three platforms.** macOS, Linux and Windows, each verified in CI.
- **Key-at-rest.** The age private key is never written to disk in plaintext — only wrapped copies exist.


Source: README.md

# Agents use secrets. They never read them.

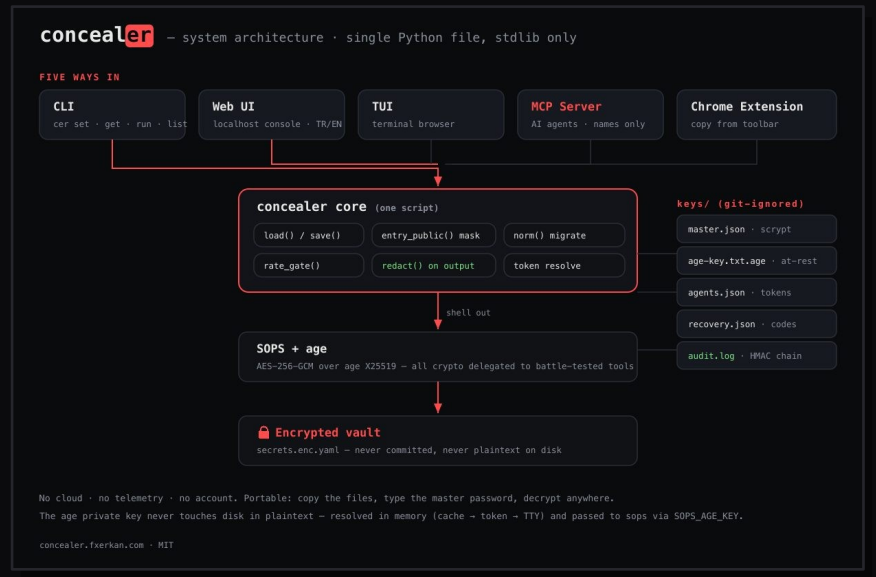
Register an agent, hand it a revocable token, and it can run psql or curl with a live credential — while the value stays out of its context entirely.

- **list_secrets** / **search_secrets** — names, types, scopes and tags. **Never values.**
- **run_with_secrets** — runs a command with the matching secrets injected into a child environment; values are **redacted from the returned output**.
- **set_secret** — writes a value, and cannot retrieve the one it just wrote.
- **Registration is mandatory.** A human token, or no token, gets *access denied*. On a hardened vault the server **fails closed**.
- **Anti-bulk-exfiltration.** Per-agent quotas: 10 rows per call, 25 distinct names per rolling hour. Set the quota to 0 to block an agent outright.
- **Every call is audited** with `source=mcp` and the agent's label as the actor.
- Works with Claude Code, Codex CLI, Gemini CLI, opencode, Cursor, Cline and Continue — same stdio server, one config line.


Source: docs/mcp.md

# One script, delegated crypto, five ways in.

- **Every operation** is load → decrypt to a dict → mutate → save → re-encrypt, shelling out to `sops` both ways.
- **The age key reaches sops in memory** via `SOPS_AGE_KEY` — never through a temp file.
- **Wrapped three ways:** by the master password (`age-key.txt.age`), by each recovery code, and by each unlock token.
- **Tokens, not prompts.** Humans get a ~8 h TTL token from `unlock`; each agent gets its own long-lived revocable token.
- **External dependencies:** `sops`, `age`, `expect` — installed for you by Homebrew, or `pywinpty` on Windows.
- **Nothing is committed:** `keys/`, `secrets.enc.yaml` and `.sops.yaml` are git-ignored. The repo ships the tool, never a vault.




Source: docs/concepts.md

# Four layers, and every ceiling written down.

Crypto

## Delegated, not invented

AES-256-GCM via SOPS over age X25519. The only crypto concealer writes itself is a stdlib scrypt password verifier and the HMAC audit chain.

Key-at-rest

## No plaintext key on disk

Only master-password-, recovery-code- and token-wrapped copies exist. A stolen folder is inert without a password or a code.

Tokens

## Revocable, client-side

The token value lives only in CONCEALER_TOKEN. The vault keeps a scrypt hash and a token-wrapped key. Revoke it and that copy is dead.

Recovery

## A real second factor

8 one-time codes. Any one recovers the vault — and passwd consumes one, so a stolen master password alone cannot take the vault over.

### Threat model — mitigation and residual risk, as documented

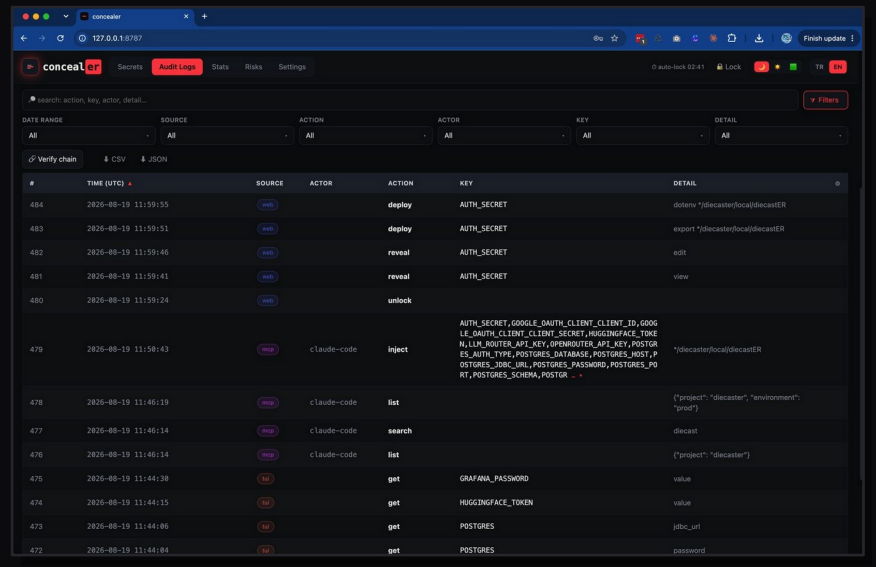
| Threat | Mitigation | Residual risk |
|---|---|---|
| Secret pasted into an AI chat | MCP injects without revealing; output redacted | Agent must be registered and rate-limited |
| Laptop stolen, disk copied | Key-at-rest — wrapped key only, no plaintext | Master password strength |
| Master password leaked | `passwd` requires a recovery code as 2nd factor | Codes must be stored separately |
| Vault committed by accident | `.gitignore` covers the vault files | User discipline on `git add` |
| Audit log tampered with | HMAC chain + `seq` + tail anchor | An FS-root attacker with the local key can re-forge |
| Agent tries to dump the vault | Registration gate + per-agent quotas | Limits are tunable per agent |

Source: docs/security.md

# Every read, write, copy and inject — chained.

One append-only log across all five interfaces. It records names and actions, never values.

- **HMAC-SHA256 chained** — each entry's hash depends on the previous one.
- **Monotonic seq** inside the signed payload, so reordering is detectable.
- **A tail anchor** (`audit.head`) catches truncation — deleting the last lines breaks verification.
- **In the Web UI:** filter by action / source / key / date, page through it, open a row, verify the chain, export CSV or JSON.
- **From the CLI:** `cer audit` and `cer audit verify`.
- **Stated ceiling:** `keys/audit.key` is local, so a filesystem-root attacker could re-forge the chain. An off-machine anchor push mitigates a full re-forge; true immutability needs an off-machine key.




Source: docs/concepts.md

# The same encrypted vault, however you like to work.

Every surface hits the same core and writes the same audit log. None of them is a bypass.

01
### CLI
Set, get, run-with and deploy from the terminal. Scriptable, CI-friendly.

02
### Web UI
A local console: type-aware forms, filters, deploy renderers, audit viewer.

03
### TUI
Three panels, keyboard-only. Browse, search, reveal and edit in the terminal.

04
### MCP
Agents list names and inject values. Registered tokens only, quota-gated.

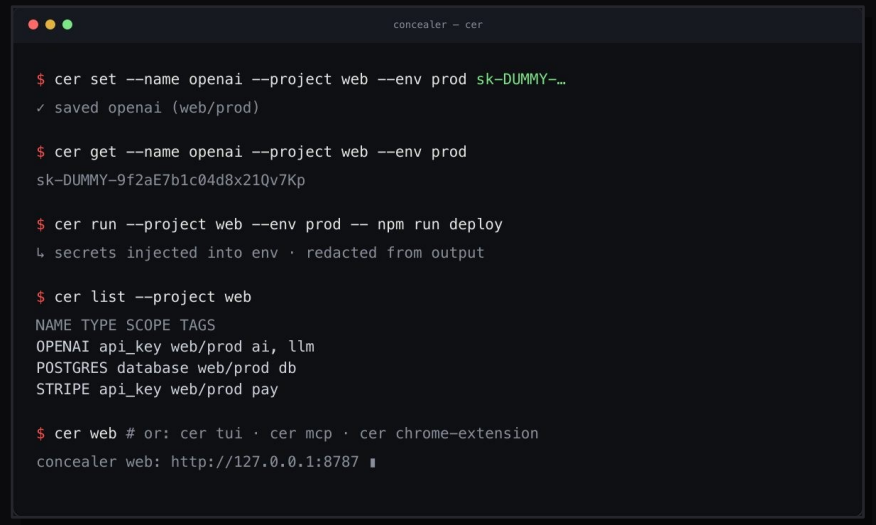
05
### Chrome
Copy a secret from the toolbar. Per-field copy, generator, three themes.

| You want… | Reach for |
|---|---|
| Scripting, CI, piping into `run` or `deploy` | CLI |
| Rich forms, filters, risk views, the audit viewer | Web UI |
| Fast keyboard browsing without leaving the terminal | TUI |
| An agent that uses secrets without seeing them | MCP |
| A password or Wi-Fi key copied while you're in the browser | Chrome extension |

# One command, any secret, nothing printed.

cer is the short alias for concealer — both names work identically. Every command takes a scope; an omitted dimension is a wildcard.

```
cer set --name OPENAI_API_KEY --project web --env prod 'sk-…' --tags ai
cer list --type database --tenant acme
cer get --name OPENAI_API_KEY --project web --env prod
cer rotate --name OPENAI_API_KEY --project web   # 32 random bytes
cer run --project web --env prod npm run deploy
cer deploy --target k8s --project web --env prod
cer scan ./myrepo --history --envvars --import
cer audit verify
```

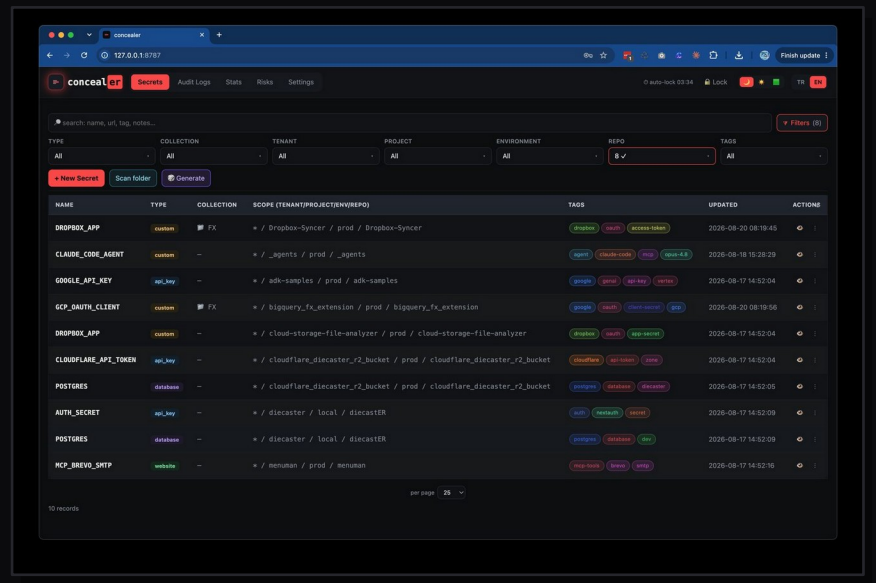



- **Scope auto-detect** — `repo` and `project` come from the current git repo when you omit them.
- **Typed injection** — a `database` secret becomes `PG_HOST`, `PG_PORT`, `PG_PASSWORD`…; an `api_key` becomes one variable.
- **8 deploy targets** — `dotenv` · `export` · `docker` · `json` · `k8s` · `aws-secrets` · `aws-ssm` · `github`. Rendered to stdout; nothing is pushed for you.
- **Guard rails** — `get`, `rm` and `rotate` refuse an ambiguous scope rather than guess.

Source: docs/cli-reference.md

# A professional console that never leaves localhost.

cer web serves a single-page app and a JSON API on 127.0.0.1 only. Single-user, unlocked with the master password.



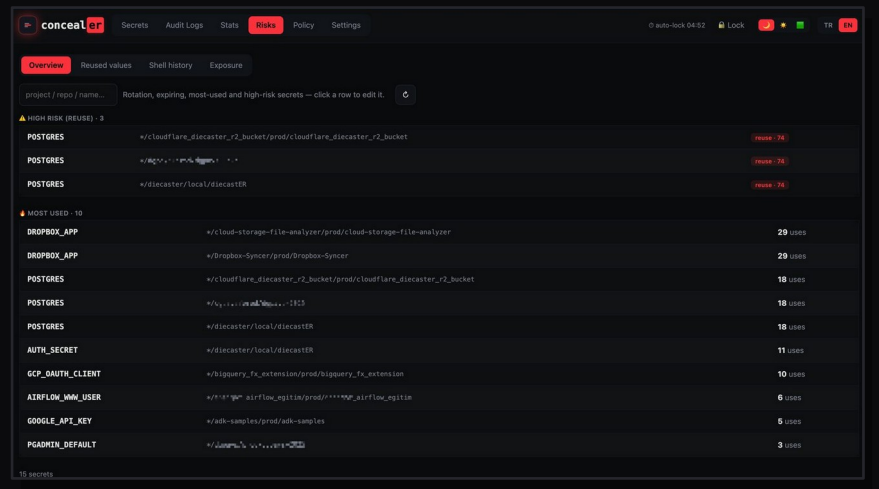

- **Bilingual TR / EN** throughout, with Dark · Light · Matrix themes and a responsive phone/tablet layout.
- **Type-aware CRUD** — each of the 15 types renders exactly the fields it needs; secret fields are stored masked.
- **Searchable multi-select filters** for type, tenant, project, environment, repo and tags; sortable and reorderable columns, including any custom field as its own column.
- **Per-secret Deploy** — render the exact CLI or manifest for eight targets, right from the row.
- **Clipboard copy with auto-clear** (20 s), show/hide toggle, and a hard **idle auto-lock** (300 s by default).
- **Risks · Policy · Scan folder · Audit · Settings** as first-class tabs.

Source: docs/web-ui.md

# Find the weak, stale, reused and already-leaked.

Four sub-views — and not one of them sends a secret value anywhere.

- **Overview** — A health dashboard: expired, expiring soon, rotation overdue, high-risk reuse, most used — each with how far past due it is.
- **Reused** — Finds the same value under more than one record and scores the blast radius. If it leaks, everything sharing it is exposed at once.
- **History** — Scans `bash` / `zsh` / `fish` history for credentials typed in the clear. Import them, then purge the lines.
- **Exposure** — HIBP Pwned Passwords via **k-anonymity** — only the first 5 hex characters of a SHA-1 leave the machine. Plus email breach lookup and a git-history / tracked-files / logs scan with a generated cleanup guide.

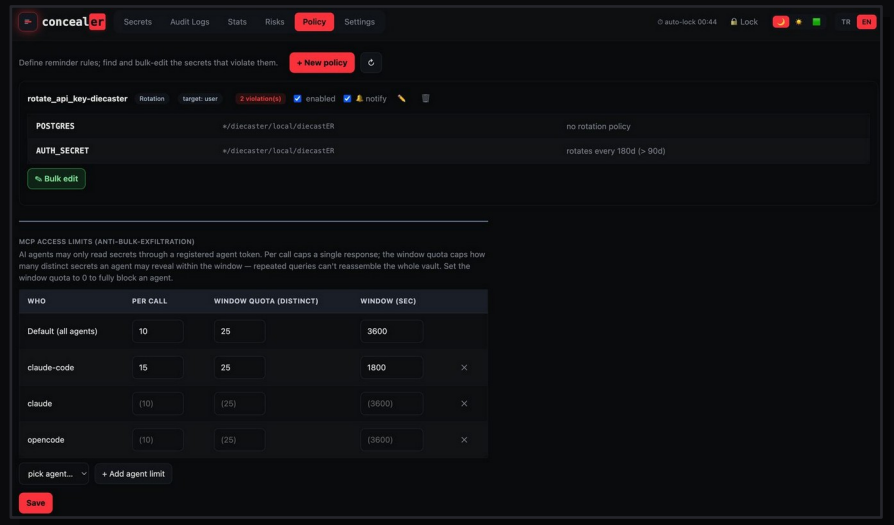



Source: docs/risks.md

# Your rules, enforced — and the agent gate.

Define reminder rules, see exactly which secrets violate them, and bulk-fix the violators. Each rule carries a target audience: user · agent · cli · web · tui · mcp · all.

| Rule kind | Flags a secret when… |
|---|---|
| Rotation | it has no rotation policy, is overdue, or its interval exceeds your maximum |
| Expiry | it is expired, expiring within N days, or missing an expiry field |
| Reuse | its value is shared with another record |
| Naming | its name doesn't match a regex you provide |
| Tagging | it is missing tags you require |



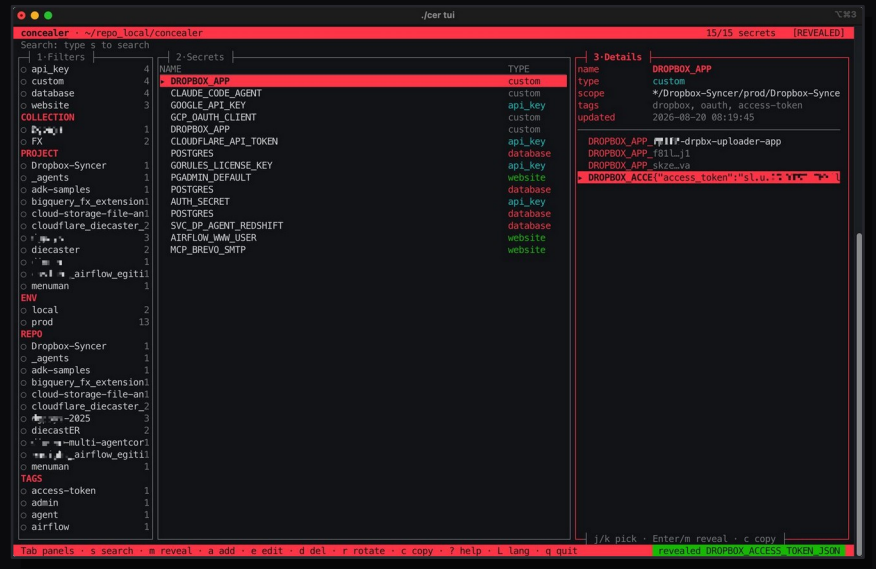

```
cer policy list
cer policy check   # non-zero exit on any violation — cron- and CI-friendly
```

Source: docs/policy.md

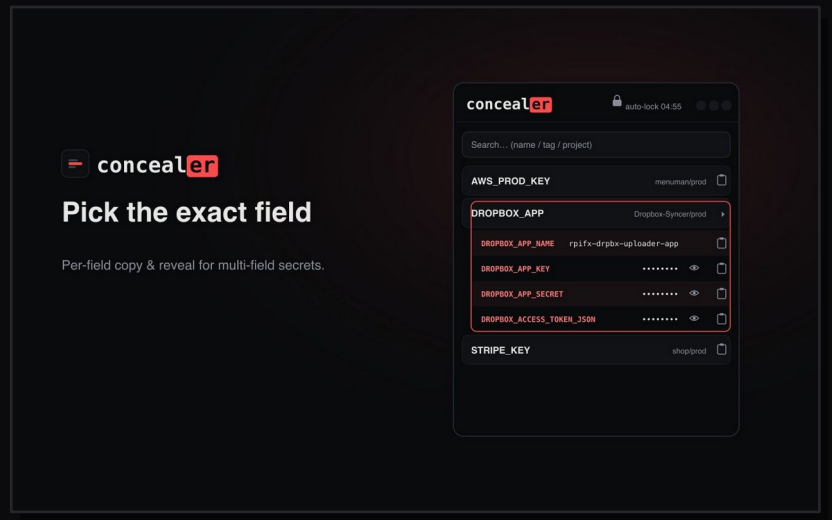
# The whole vault, keyboard only.

Three panels — Filters, Secrets, Details. Browse, search, reveal, copy and edit without leaving the shell.

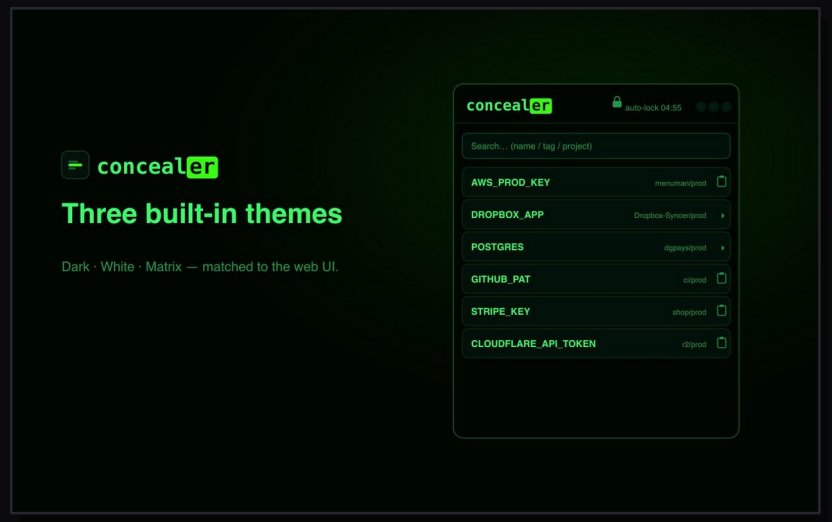
- **Vim-style navigation** — `j`/`k`, `g`/`G`, `Tab` between panels, `1`/`2`/`3` to jump.
- **Live search** with `s` or `/`; `x` clears every filter.
- **Reveal per field** — `m` on a record, or pick a single field in Details. Copy value, username or url with `c` / `u` / `w`.
- **Clipboard auto-clears after 45 s.** There is no paste — to put a value in, use add/edit.
- **Bilingual** — `L` toggles TR/EN, `?` shows the shortcut table in-app.
- **Not a bypass** — every reveal, create, update and delete is audited with its source.




Source: docs/tui.md




Per-field copy for multi-field secrets.



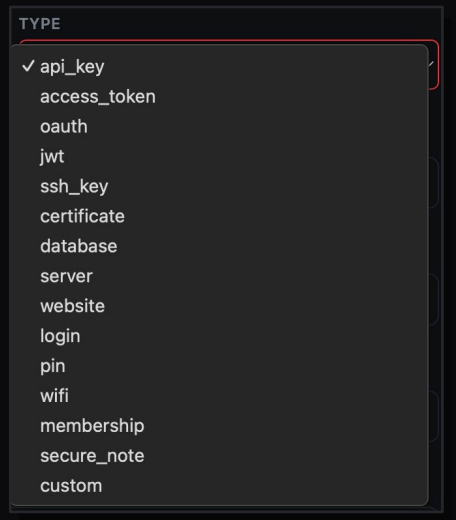

Matrix theme.

Source: docs/chrome-extension.md

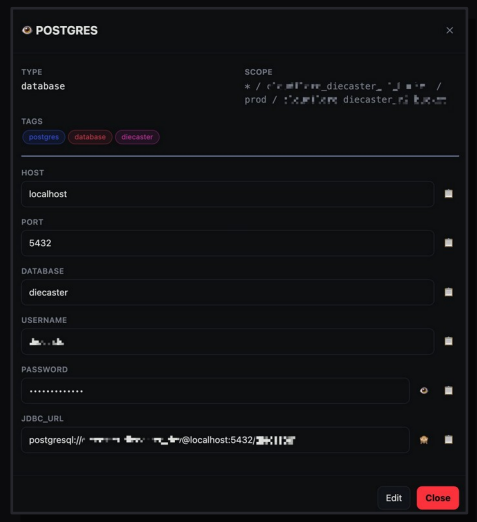
# Copy a secret from the toolbar.

A thin popover over your local vault. It talks only to 127.0.0.1 — no cloud, no account, no telemetry. On the Chrome Web Store.

- **Per-field copy** — multi-field secrets expand into child rows; copy or reveal exactly the field you need.
- **On-demand server** — the popup starts `concealer web` when you open it and self-exits after 15 min idle. Nothing lingers.
- **Auto-lock countdown** in the header, blinking as it runs out; clipboard auto-clears; values are never logged.
- 🎲 **Generator** — strong passwords, hex, base64url, UUID. Works before you unlock.
- **Three themes** — Dark · White · Matrix, matching the web console.
- **The native helper ships inside concealer** (`concealer native-host`) — `cer chrome-extension` just registers it, once per machine.




The type picker.




A database secret's form.

# Typed secrets, four-dimension scope.

A credential is not a string. Each type renders exactly the inputs it needs, and secret fields are stored masked and revealed only on demand — audited.

- **15 types** — `api_key` · `access_token` · `oauth` · `jwt` · `ssh_key` · `certificate` · `database` · `server` · `website` · `login` · `pin` · `wifi` · `membership` · `secure_note` · `custom`.
- **Masking is record-aware**, resolved in order: per-field override → type template → field-name heuristic → value heuristic (a `scheme://user:pass@host` DSN is masked even in a plain field).
- **Scope: tenant / project / environment / repo.** An empty dimension is a wildcard; the most-specific match wins at inject time.

- **No PII by design** — there are deliberately no credit-card, passport or national-ID types. This is a machine-and-account credential vault.

Source: docs/secret-types.md

Persona 01

# DevOps / Platform Engineer

Owns the pipelines and the blast radius. Needs secrets out of .env files, out of shell history, out of CI logs — and a gate that fails the build when they creep back in.

## The pain today

- A dozen repos each carrying their own **.env**, all with a key called `OPENAI_API_KEY` and no way to tell them apart.
- **export AWS_SECRET_ACCESS_KEY=…** sitting in `~/.zsh_history` in the clear.
- A key was committed six months ago and nobody knows which repos still carry it.
- No record of who read which credential, or when.
- New laptop = chasing credentials in Slack threads.

## What they use

- **cer run** — injects the scoped secrets into a child process and redacts them from its output. Nothing lands in the shell.
- **cer deploy --target** — renders to `dotenv · export · docker · k8s · json · aws-secrets · aws-ssm · github`. Nothing is pushed; you pipe it.
- **cer scan** — sweeps a folder, shell history and live env vars, then imports what it finds tagged by origin.
- **cer policy check** — exits non-zero on any violation. Drop it in cron, a pre-commit hook or CI.
- **cer gitscan** — pickaxes git history and tracked files for vault values, then writes a cleanup guide (it never rewrites history for you).
- **cer audit verify** — recomputes the HMAC chain and the tail anchor.

## What they get

- Values never reach the terminal, the transcript or the log.
- Scope is auto-detected from the git repo — `--repo` and `--project` fill themselves in.
- Most-specific scope wins at inject time: `acme/proj-a/prod` beats `proj-a` beats global.
- Zero infrastructure to run, patch or pay for.
- The vault is an encrypted SOPS file — it diffs and versions next to the code.

Persona 02

# Cloud Architect / Solution Architect

Runs demos, PoCs and pilots across many customers and three clouds. Has to defend the design in a security review — and hand the whole thing over without standing up a server.

## The pain today

- Customer credentials for a PoC mixed in with personal and internal keys.
- Three clouds, a dozen tenants, four environments — one flat namespace can't express that.
- A security review asks "where does this credential go?" and the honest answer is a shrug.
- Standing up Vault for a two-week PoC is absurd; a SaaS vault means procurement and a DPA.
- The vault is tied to one laptop's Keychain, so the handover is a rebuild.

## What they use

- **tenant / project / environment / repo** as the taxonomy — the same `DATABASE_URL` exists once per customer and stage without name mangling.
- The **Web UI** as the console: type-aware forms, multi-select filters, sortable columns, per-secret deploy renderers.
- The **threat-model table** and **documented ceilings** as review artefacts — including what the tool does *not* protect.
- **cer export / import** — one password-protected `.cerbak` bundle to hand to the customer.
- **Portability**: copy `secrets.enc.yaml` + `keys/`, type the master password on the new machine. No TPM, no Keychain, no re-key.
- The **comparison matrix** for the build-vs-buy conversation.

## What they get

- No server, no container, no cloud tenant, no procurement — `init` and there is a vault.
- Crypto is delegated to **SOPS** (CNCF) and **age**: AES-256-GCM over X25519. Nothing home-grown.
- One readable MIT-licensed Python script a customer's security team can actually read.
- A copied folder is **inert** without the master password or a recovery code.
- No PII types by design — this is a machine-credential vault, not an identity store.

`docs/security.md` · `docs/portability.md` · `docs/comparison.md`

Persona 03

# End User

Not a DevOps engineer. Has Wi-Fi passwords, door PINs, logins and a couple of API keys — and wants none of it in a notes app, a browser profile or someone else's cloud.

## The pain today

- Passwords in a notes file, a spreadsheet, or the browser's password manager.
- The same password reused across half a dozen sites, with no idea which ones have leaked.
- Another $3–12 a month subscription, and an account on someone else's server.
- Forgetting the master password means losing everything.

## What they use

- The **Chrome extension**: click the toolbar icon, unlock, copy. Multi-field secrets expand so you copy exactly the field you need.
- The **Web UI** at `127.0.0.1:8787` — fully bilingual TR/EN, with forms for `wifi · pin · login · website · membership · secure_note`.
- 🎲 **Generate** for strong passwords, hex, base64url or a UUID — one click to copy.
- **Risks → Exposure**: check a value against HIBP Pwned Passwords, and an address against HIBP breaches.
- **8 recovery codes** printed at setup — any one of them gets you back in.

## What they get

- No account, no subscription, no sync server. Nothing leaves the machine.
- Clipboard auto-clears after 20 s; the session auto-locks on idle (300 s by default).
- Leak checks use **k-anonymity** — only the first 5 hex characters of a SHA-1 ever leave the box.
- Three themes (Dark · White · Matrix) and a Turkish interface throughout.
- Honest limits: no mobile app and no browser autofill — the copy button is the workflow.

# Local-only, zero-infra, agent-native — the gap the other three markets leave open.

✓ yes ~ partial × no ★ only concealer

| Capability | concealer | 1Password / Bitwarden | HashiCorp Vault | Doppler / Infisical | SOPS + age (raw) | pass / gopass | secretctl |
|---|---|---|---|---|---|---|---|
| Deployment | Local, single file | SaaS | Self-host / HCP | SaaS (or self-host) | Local | Local | Local, single binary |
| Server or daemon needed | × none | cloud | ✓ server | ✓ | × | × | × none |
| Cost | Free (MIT) | ~$3–12 / user / mo | Free OSS / $$$ ent. | paid tiers | Free | Free | Free |
| Git-versionable vault | ✓ values encrypted | × | ~ | × | ✓ | ✓ | × SQLite blob |
| Typed secrets (db / api / ssh…) ★ | ✓ templates | ~ item types | × | × | × | × | — |
| 4-dimension scoping | ✓ tenant/proj/env/repo | ~ vaults, tags | ✓ paths | ✓ envs | × | ~ dirs | ~ wildcards |
| Tamper-evident audit log | ✓ HMAC + anchor | ~ cloud logs | ✓ | ✓ | × | ~ git log | ✓ HMAC |
| MCP-native for AI agents | ✓ gate + quota | × | ~ SDK | ~ SDK | × | × | ✓ MCP |
| Per-agent exfiltration quotas ★ | ✓ | × | ~ policy | × | × | × | — |
| Works fully offline | ✓ | ~ cache | ~ | × | ✓ | ✓ | ✓ |
| Dynamic / leased secrets | × | × | ✓ | ~ | × | × | × |
| Multi-user / RBAC / SSO | × single-owner | ✓ | ✓ | ✓ | ~ recipients | ~ keys | × |
| Mobile app / autofill | × | ✓ | × | × | × | ~ | × |

Source: docs/comparison.md

# Where concealer wins — and where you should use something else.

## Where it wins

- **Zero infrastructure.** No server, container, cloud tenant or daemon. `init`, and you have a vault.
- **Git-native.** An encrypted YAML file that diffs and versions in the same repo as the code.
- **Agent-first.** The only tool in the matrix with an MCP server designed around AI-agent threat models: registration gate, per-agent exfiltration quotas, redacted output.
- **Tamper-evident by design.** HMAC chain plus a head anchor that catches tail-truncation — no cloud audit pipeline required.
- **No lock-in, no telemetry, one file to read.** Compare with trusting a proprietary cloud vault — see the 2022 LastPass breach.

## Where it doesn't — use something else

- **Teams.** No SSO, no RBAC, no per-user sharing. It is a single-owner vault. → 1Password / Bitwarden / Vault.
- **Dynamic secrets and leases.** No short-lived DB credentials minted on demand. → HashiCorp Vault.
- **Automatic rotation and a CI/CD sync fabric.** Rotation is manual. → Doppler / Infisical.
- **Consumer UX.** No mobile app, no browser autofill, no passkeys. → 1Password / Bitwarden / Keeper.
- **Turnkey compliance attestations.** No FedRAMP or SOC 2 — those are earned by the entity operating a tool, not shipped by the tool. → Keeper / Vault / cloud KMS.

| If you need… | Reach for |
|---|---|
| A personal or single-dev vault with no server, versioned in git | **concealer** |
| Safe secret access for local AI agents and MCP clients | **concealer** |
| To just encrypt a config file in a repo | **concealer** |
| Team sharing, SSO, mobile autofill | 1Password / Bitwarden |
| Dynamic DB credentials, leases, encryption-as-a-service | HashiCorp Vault |
| Managed multi-environment secrets synced into CI/CD | Doppler / Infisical |

Source: docs/comparison.md

# Up and running in about a minute.

Pick your package manager — each one pulls in the sops and age binaries concealer wraps. Then init once: it prints 8 recovery codes and a starter token, and removes the plaintext age key from disk.

### macOS · Homebrew

```
# pulls in sops, age, expect
brew install fxerkan/tap/concealer

concealer init
eval "$(cer unlock)"
```

### Linux · pipx

```
pipx install concealer
sudo apt install sops age

concealer init
eval "$(cer unlock)"
```

### Windows · Scoop

```
scoop bucket add fxerkan \
  https://github.com/fxerkan/scoop-bucket
scoop install concealer

concealer init
$env:CONCEALER_TOKEN = "…"
```

Source: docs/installation.md

# conceal er

# Stop pasting keys into chat windows.

Local-only. Portable. Open source. Built for the way we code now.

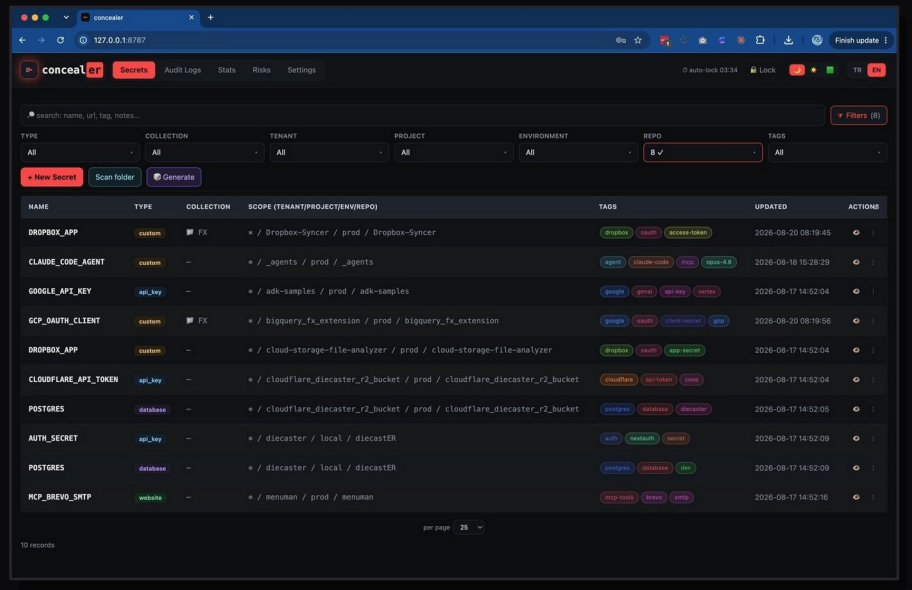
```
brew install fxerkan/tap/concealer
```

```
pipx install concealer
```

```
scoop install concealer
```

MIT

macOS · Linux · Windows




- concealer.fxerkan.com
- GitHub
- Full comparison
- Chrome Web Store
- Support

---

concealer v0.9.24 · MIT · Developed by FXerkan — Code more, worry less.