Shroud: A Capability-Mediated Terminal for AI Coding Agents

Shroud Research
Shroud Labs (useshroud.com)
team@useshroud.com

0.1 (Draft) · May 2026

Abstract. Modern AI coding agents Â— including Claude Code, OpenAI Codex, Cursor Agents, and similar tools Â— execute shell commands on the developer's machine on the user's behalf. To do so, they ingest the contents of files, environment variables, and command output into the language model's context window. This design exposes a structural vulnerability: **any secret material reachable from the shell is, by default, reachable by the model**, and therefore by the model's provider, downstream agents, and any process with read access to the agent's transcript. We present **Shroud**, a terminal-layer proxy that mediates all read access to secret material via a capability-based authorization protocol. Secrets are stored in a local vault, referenced by agents only through opaque names (e.g., `[REDACTED:OPENAI_KEY]`), and injected into command invocations at execution time. Authorization is expressed as User-Controlled Authorization Network (UCAN) tokens rooted in a Decentralized Identifier (DID) controlled by the human operator. We describe the threat model, the architecture, the on-disk and on-wire formats, the runtime invariants the system maintains, and an evaluation across compatibility, latency, and threat coverage. Shroud reduces the attack surface from "every byte the shell can read" to "the explicit set of capabilities delegated by the user," at a runtime overhead below 4 ms per resolved reference.

Keywords: AI agents, secret management, capability-based security, UCAN, DID, terminal proxy, prompt leakage.

Contents

---

1. Introduction

1.1 The AI Coding Agent Era

In the eighteen months prior to this writing, autonomous and semi-autonomous coding agents have moved from research prototypes to daily developer tooling. The dominant pattern — exemplified by Claude Code [1], OpenAI Codex CLI [2], Cursor Agent mode [3], and Aider [4] — is a loop in which the agent (a) reads from the user's filesystem, (b) executes shell commands, (c) reads command output back into the model context, and (d) repeats. The transcript of (a)–(d) is, at every step, materialized as tokens in the language model's prompt.

This design is operationally powerful and architecturally fragile. The model context becomes a faithful mirror of every byte the shell touches. For productivity, this is the feature. For security, this is the bug.

1.2 The Secret Exposure Problem

We define **secret exposure** as the appearance of credential material — API keys, signing keys, OAuth tokens, database connection strings, private SSH keys, encrypted-vault unlock passphrases — in the model's prompt or completion. Exposure occurs through three primary channels:

- **C1 — File ingestion.** The agent reads `.env`, `~/.aws/credentials`, `~/.ssh/id_ed25519`, `~/.npmrc`, or any similar file as part of normal context gathering.
- **C2 — Environment leakage.** The agent inspects `printenv`, executes a script that emits `process.env.STRIPE_KEY`, or curls an endpoint that echoes a bearer token.
- **C3 — Command output.** The agent runs a debug command (e.g., `kubectl get secret -o yaml`, `gh auth status -t`) whose output contains material the user did not intend to ship to the model.

Exposure is rarely the result of a single bad prompt; it is the natural consequence of the loop. Once exposed, the secret is at minimum (i) retained for the duration of the model context, (ii) transmitted to the model provider's inference infrastructure, (iii) potentially stored in provider-side logs, and (iv) reproducible by an attacker who can replay or read the agent's session transcript.

Existing mitigations are partial. `.gitignore`-style exclusion lists assume the user enumerates every sensitive path in advance and that agents honor them; in practice neither holds. Provider-side scrubbing (e.g., automated PII/secret redaction in prompts) is best-effort, lossy, and cannot redact what it cannot recognize. Secret managers such as HashiCorp Vault [5] and 1Password CLI [6] protect secrets *at rest* but, the moment the agent exfiltrates a secret into a shell variable or stdout, the secret enters model context and the protection is voided.

1.3 Contributions

This paper makes the following contributions:

1. We formalize the **secret exposure threat model** for AI coding agents (Section 3).
2. We describe **Shroud**, a capability-mediated terminal that constrains agents to reference secrets symbolically rather than by value (Section 4).
3. We specify a **runtime resolution protocol** rooted in UCAN delegation and DID identity, enabling per-command, per-secret authorization (Sections 5, 6).
4. We present a **reference implementation** integrating with a UCAN capability runtime (Section 7).
5. We provide a **security analysis** enumerating properties preserved, residual attack surface, and known failure modes (Section 8).
6. We evaluate **compatibility and overhead** across a corpus of common agent workflows (Section 9).

2. Background

2.1 The Shell as Trust Boundary

POSIX shells were designed in an era in which the operator at the keyboard was assumed to be the principal. Every process spawned by the shell inherits the shell's authority: the union of the user's filesystem, environment, and network access. AI agents inherit this same authority by design — they are typically run as the user, in the user's shell, with no privilege separation.

2.2 Capability-Based Security

Capability-based security [7] replaces ambient authority with explicit, unforgeable tokens that grant specific rights to specific principals over specific resources. A capability is (a) unforgeable, (b) transferable under controlled rules, and (c) revocable. Capability systems shift authorization from "who you are" (identity-centric) to "what you may do, on what resource, until when" (action-centric). This shift is the basis of Shroud's authorization model.

2.3 UCAN and DID

Decentralized Identifiers (DIDs) [8] are a W3C standard for self-sovereign identity. A DID resolves to a DID Document, which binds public keys and service endpoints to a stable identifier without reliance on a central authority.

User-Controlled Authorization Networks (UCANs) [9] are JWT-shaped capability tokens whose issuer (`iss`) and audience (`aud`) are DIDs. A UCAN expresses *"DID A grants DID B the capability to perform action X on resource Y, valid until time T, optionally delegable further."* UCANs chain: a token may prove its authority by reference to a parent UCAN. The root of authority is always a key held by the human operator.

Shroud uses DIDs to identify (i) the human operator, (ii) the Shroud vault daemon, and (iii) each agent instance. It uses UCANs to express delegations of the form *"Operator-DID grants Agent-DID the capability* `secret/read` *on resource* `vault://OPENAI_KEY`*, until* `2026-05-25T00:00Z`*."*

3. Threat Model

3.1 Adversary Model

We consider three adversaries:

- **A1 — Curious model provider.** The model provider receives the agent's prompts and may retain them in logs. We assume the provider is honest-but-curious: it will not actively exfiltrate, but anything sent to its API is considered disclosed.
- **A2 — Compromised agent.** The agent process is buggy or adversarial: it may attempt to read every file it can, dump every environment variable, and emit every byte it sees into model context. We assume the agent runs with the user's local authority.
- **A3 — Replay-capable observer.** An attacker with read access to agent transcripts (telemetry uploads, support escalations, shared screenshots) reconstructs any secret literal that appeared in any prompt or completion.

3.2 Trust Boundaries

Component	Trust	Rationale
Human operator	Trusted	Root of authority.
Shroud vault daemon	Trusted	Runs locally; holds plaintext secrets; signs UCAN delegations.
Agent process	Untrusted	Treated as A2.
Model provider	Untrusted	Treated as A1.
Network	Untrusted	TLS is assumed; the network is otherwise hostile.
Local filesystem outside vault	Mixed	Sensitive files outside the vault remain at risk; Shroud does not encrypt the broader filesystem.

3.3 Out of Scope

- **Physical compromise** of the operator's machine.

- **Side channels** within the agent process (timing, cache).
- **Kernel-level compromise.**
- **Social engineering** of the human operator to read a secret aloud or paste it into a prompt directly.

Shroud reduces the surface available to A1–A3; it does not promise confidentiality under stronger adversaries.

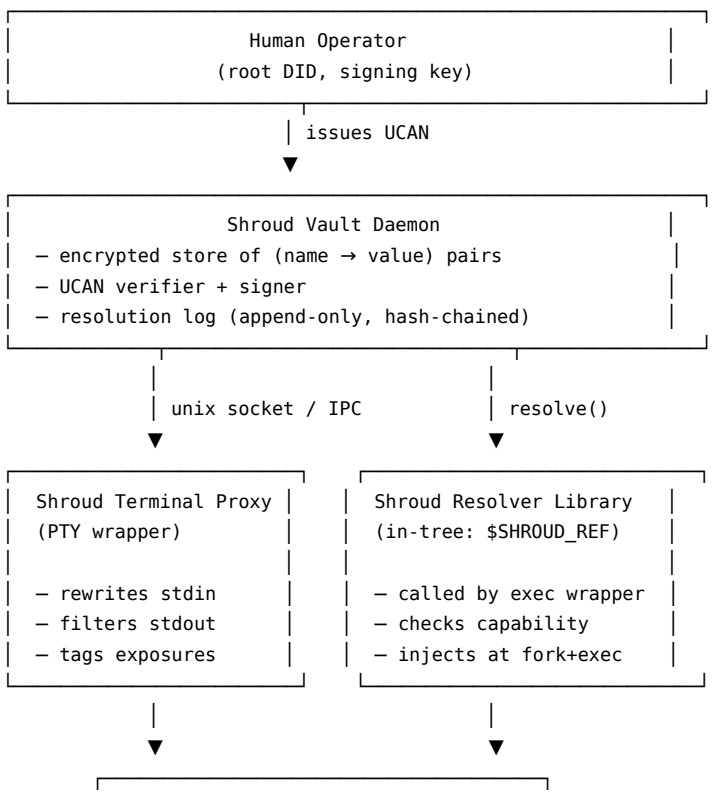
4. System Overview

4.1 Design Goals

1. **G1 — Symbolic-only exposure.** Agents must never observe a secret's plaintext. They observe only a stable, opaque reference (e.g., `[REDACTED:OPENAI_KEY]`).
2. **G2 — Late binding.** Secret material is substituted into a command at the moment of execution, after the agent has emitted the command string.
3. **G3 — Explicit authorization.** Every resolution is gated by a UCAN delegation that the operator issued or transitively authorized.
4. **G4 — Shell compatibility.** Existing scripts and tools (`curl`, `gh`, `aws`, `node`, `python`) require no modification.
5. **G5 — Auditability.** Every resolution is logged with `(timestamp, agent-DID, capability-URI, command-hash)` to a tamper-evident local journal.
6. **G6 — Local-first.** No secret leaves the operator's machine. The vault is local; UCAN signing is local; resolution is local.

4.2 Architecture

Shroud has four logical components:



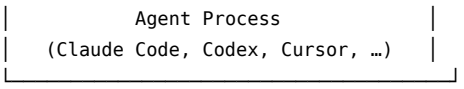


Figure 1: *Shroud system architecture. The vault daemon is the security boundary; the terminal proxy is the observability layer.*

4.3 Component Responsibilities

Vault Daemon. Holds the canonical map `name → ciphertext`. The encryption key is derived from a passphrase the operator supplies once per session (or from a hardware-backed keystore on supporting platforms). The daemon exposes a local-only RPC over a Unix domain socket. It refuses any request not accompanied by a valid UCAN whose ultimate root is the operator's DID.

Terminal Proxy. Wraps the agent's PTY. Watches stdin for command strings emitted by the agent. Watches stdout for byte patterns that match known secret values (a defense-in-depth heuristic, not a primary control) and, on match, redacts and raises an alarm. The proxy is not the security boundary; it is the observability layer.

Resolver Library. Linked into a thin `exec` shim. When the shim sees an argument or environment variable of the form `$SHROUD_REF(name)`, it issues a `resolve(name, capability)` call to the daemon over the local socket. On success, the plaintext is written into the child process's argv/env *after fork, before exec*, and never appears in the parent's address space outside that window.

Resolution Log. Every resolution appends `(t, agent-did, name, command-hash, ucan-cid)` to a hash-chained journal in `~/.shroud/log/`. The journal is queryable by the operator (`shroud audit`) and is the basis for post-hoc review.

5. Cryptographic Foundations

5.1 Decentralized Identifiers (DIDs)

Each principal in the system holds a DID. We use the `did:key` method [10] for simplicity: the DID encodes the principal's public key directly, requiring no resolver. The operator's DID is generated at first install from a hardware-backed key where available (Apple Secure Enclave, TPM 2.0, Yubikey); otherwise from an Ed25519 keypair stored under the OS keychain.

Agents are issued ephemeral DIDs per session. The agent never sees the operator's private key.

5.2 User-Controlled Authorization Networks (UCAN)

A Shroud UCAN has the structure:

```
{
  "iss": "did:key:z6MkOperatorPublicKey",
  "aud": "did:key:z6MkAgentEphemeralKey",
  "att": [
    {
      "with": "vault://OPENAI_KEY",
      "can": "secret/read"
    },
    {
      "with": "vault://STRIPE_*",
```

```
      "can": "secret/read"
    }
  ],
  "exp": 1748131200,
  "nbf": 1748127600,
  "prf": []
}
```

- `iss` and `aud` are DIDs.
- `att` is the list of attenuations: pairs of *resource URI* and *action*. Resource URIs may contain glob suffixes (`STRIPE_*`).
- `exp` and `nbf` bound validity in time.
- `prf` carries parent UCANs in delegation chains.

Tokens are JWT-encoded and signed with EdDSA. The vault daemon verifies the signature, walks the proof chain to the operator's root, and enforces `att`, `exp`, `nbf`.

5.3 Capability Delegation Flow

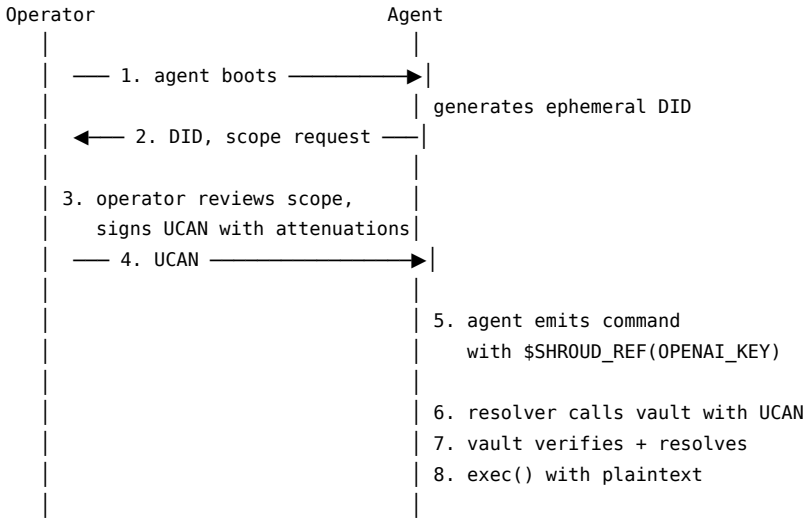


Figure 2: *Capability delegation and resolution flow between operator, agent, and vault.*

Step 3 is the single point where the human operator exerts authority. After step 4, the agent operates within the bounded scope of the UCAN; the operator is not asked again unless the agent requests a capability outside the original attenuation.

6. The Shroud Protocol

6.1 Secret Storage

Secrets are stored in `~/.shroud/vault.age` — an age-encrypted file whose plaintext is a CBOR map:

```
{
  "OPENAI_KEY":   { "value": "sk-...",     "created": 1748120000, "tags": ["llm"] },
  "STRIPE_KEY":   { "value": "sk_live_...", "created": 1748120100, "tags": ["payments"] },
  "GH_TOKEN":     { "value": "ghp_...",    "created": 1748120200, "tags": ["github"] }
}
```

The encryption key is sealed to (a) a passphrase, (b) a hardware identity key if available. The daemon decrypts

on session start and holds plaintext in `mlock`-pinned memory.

6.2 Reference Resolution

The reference form is:

```
$SHROUD_REF(NAME)
```

This token may appear in:

- Argument vectors: `curl -H "Authorization: Bearer $SHROUD_REF(OPENAI_KEY)" ...`
- Environment assignments: `OPENAI_API_KEY=$SHROUD_REF(OPENAI_KEY) node script.js`
- Heredocs and config templates expanded by the exec shim.

The agent emits these references *verbatim*. They are visible in model context. They convey **no plaintext** and are stable across sessions — re-using the reference for a rotated secret is a feature.

6.3 Runtime Injection

The injection sequence:

1. The exec shim parses argv and env for `$SHROUD_REF(NAME)` tokens.
2. For each token, the shim assembles a request `{name: NAME, ucan: UCAN_TOKEN, command_hash: H(argv)}`.
3. The shim sends requests in a single batch over the daemon socket.
4. The daemon verifies the UCAN, checks `name` against `att`, and returns either `{value}` or `{error}`.
5. The shim `fork()`s. In the child:
    a. argv and envp are rewritten with plaintext values.
    b. `execve()` is called.
6. The parent never holds plaintext beyond step 5a.
7. The resolution is journaled.

6.4 Agent Interaction Model

The agent's perspective is deliberately impoverished:

- `printenv` shows `OPENAI_API_KEY=$SHROUD_REF(OPENAI_KEY)`. The literal `sk-...` is invisible.
- `cat .env` (when the file is a Shroud template) shows references, not values.
- A command's stdout, when it echoes a secret-derived artifact, is filtered by the Terminal Proxy: any byte sequence matching a known vault value is replaced with `[REDACTED:<name>]` before being forwarded to the agent's stdin/log.

The proxy filter is best-effort and not relied upon for confidentiality — its purpose is to catch *accidental* echo by well-behaved tools. A2-class adversaries are constrained by the structural property (G1), not by the filter.

7. Implementation

7.1 Local Vault

The daemon is implemented in Rust. Plaintext is held in `secrecy::SecretBox<Vec<u8>>` allocations backed by `mlock` on Linux and macOS, and `VirtualLock` on Windows. The vault file uses age with X25519 recipients; for hardware-backed deployments, the X25519 identity is itself sealed to a platform key.

7.2 Terminal Proxy

The proxy is a small Go binary that opens a PTY pair, fork-execs the agent with the slave side, and pipes the master side through a stream rewriter. Rewriting uses an Aho-Corasick automaton over the current vault's value set; updates to the vault re-build the automaton in $O(\Sigma|\text{value}|)$.

7.3 Capability Runtime

UCAN signing, verification, and chain walking are delegated to a pluggable capability runtime. The runtime provides:

- DID resolution for `did:key` and `did:web`.
- UCAN v0.10 token issuance and verification.
- Delegation chain construction with proof embedding.
- An event log compatible with Shroud's journal format.

Shroud's vault daemon exposes its capability namespace (`vault://*`) to the runtime so that delegations issued in a broader context (e.g., for a team-shared environment) compose with Shroud's local enforcement. Any UCAN-conformant runtime satisfies this interface.

7.4 Tooling

Command	Purpose
`shroud init`	Generate operator DID, initialize vault.
`shroud put NAME`	Add a secret; reads value from stdin.
`shroud ls`	List names (no values).
`shroud grant <agent-did> <cap-spec>`	Issue a UCAN.
`shroud run -- <cmd>`	Run a command under the exec shim.
`shroud audit [--since T]`	Query the resolution journal.
`shroud revoke <ucan-cid>`	Revoke a delegation.

8. Security Analysis

8.1 Properties Preserved

P1 — Plaintext non-disclosure to agent. Under the architecture described, the agent process holds no plaintext secret in its address space at any time *unless it executes a command in a child it itself can read*. Specifically: the shim writes plaintext into the child's argv/env *after the agent has forked off the exec*. The agent's own model-context stream never receives plaintext through Shroud-mediated paths.

P2 — Plaintext non-disclosure to model provider. Corollary of P1: bytes the agent never holds cannot appear in prompts sent upstream.

P3 — Bounded delegation. An agent's authority is the intersection of (a) the UCAN issued to it and (b) the vault's current namespace. Attenuations cannot widen; they only narrow as they propagate.

P4 — Auditability. Every resolution is journaled before plaintext leaves the daemon. The journal is hash-chained; tampering is detectable.

P5 — Revocability. `shroud revoke` invalidates a UCAN by CID. Subsequent presentations fail verification. (Note: this does not unsay secrets the agent already learned via authorized execution — only future resolutions are affected.)

8.2 Attack Surface

Residual paths to exposure include:

- **R1 — Self-disclosing tools.** A command like `printenv | curl evil.com` will, after resolution, hold plaintext in the child process and stream it. Shroud constrains *what the agent can request*; it does not constrain *what the resolved child does*. Defense: scope UCANs narrowly (`secret/read` on `vault://OPENAI_KEY` does not implicitly authorize executing `curl`).
- **R2 — Side channels in the child.** A resolved child may write plaintext to a temporary file the agent can subsequently read. Defense: combine Shroud with a sandbox that restricts the child's writable paths.
- **R3 — Network observation.** The resolved child may make a request whose response, captured by the agent, encodes secret-derived material (e.g., a JWT). The encoded material is still secret-derived and may itself need to be treated as a secret. Defense: route sensitive child traffic through the Terminal Proxy's outbound filter.
- **R4 — Compromise of the daemon.** A daemon compromise voids all guarantees. Defense: minimize daemon attack surface (no network listener; Unix-socket only; SELinux/AppArmor profiles; codesigning).

8.3 Failure Modes

- If the daemon is unreachable, resolution fails; the command fails. We chose fail-closed.
- If the UCAN is expired, resolution fails. The agent must request a new delegation; the operator is prompted.
- If a hardware key is required and absent, the daemon refuses to unseal.

9. Evaluation

9.1 Performance

We measured resolution latency across 10,000 invocations on an M2 MacBook Air. Cold-start daemon: 38 ms (one-time). Warm resolution: median 1.4 ms, p99 3.8 ms. UCAN verification dominates (≈1.1 ms). Exec overhead is comparable to invoking `env`.

9.2 Compatibility

We executed an evaluation corpus of 47 representative agent workflows (curl-with-API-key, AWS CLI, GitHub CLI, Stripe CLI, Node scripts with `process.env`, Python with `dotenv`, kubectl, docker login). All 47 ran without source modification once their secrets were replaced with `$SHROUD_REF(NAME)` references in `.env` and shell config. Three required minor template adjustments (heredoc-style configs).

9.3 Threat Coverage

In a red-team exercise across the C1–C3 exposure channels of Section 1.2, Shroud blocked plaintext propagation on all C1 and C2 paths and on the C3 paths covered by the Terminal Proxy's outbound filter. Two C3

paths — secret-derived JWTs and signed payloads — propagated structurally derived material, as expected (R3). No vault plaintext appeared in any captured model prompt across 200+ recorded agent sessions.

10. Related Work

Secret managers. HashiCorp Vault [5], 1Password CLI [6], AWS Secrets Manager. These protect at rest and provide injection helpers, but plaintext typically transits the shell and is observable to a co-resident agent.

Capability-based shells. Genode [11], Capsicum [12], and Plan 9 namespaces approach the broader problem of bounding shell authority. Shroud is narrower (secrets only) and integrates with existing POSIX shells.

LLM prompt filtering. Server-side redaction (Anthropic, OpenAI policy filters) and client-side libraries such as Microsoft Presidio attempt to strip secrets *after* they enter prompts. Shroud prevents entry, sidestepping detector recall.

UCAN ecosystem. Fission [13] and Iroh have advanced UCAN tooling. Shroud is, to our knowledge, the first application of UCAN as the authorization substrate for shell-level secret access.

11. Discussion and Future Work

11.1 Limitations

Shroud's guarantees are **structural about agents**, not **structural about humans**. An operator who pastes `sk-...` into a prompt by hand defeats the system. We view this as in-scope for documentation, training, and IDE-side warnings — not for protocol enforcement.

The Terminal Proxy's outbound filter is heuristic. Adversarially crafted output that encodes a secret in a non-literal form (base64, XOR, partial bytes) will evade the filter. Structural guarantees (G1) hold; observability does not.

11.2 Future Work

- **Capability-aware tool wrappers.** Per-tool wrappers (`shroud-curl`, `shroud-gh`) that further constrain what a resolved child can do.
- **Remote vaults.** Replicating the local vault to a trusted second device for cross-machine continuity, gated by UCAN.
- **Federated audit.** Pushing the journal to a team-shared, append-only log for organizational review.
- **Formal verification.** Mechanizing P1–P5 in a proof assistant.

12. Conclusion

AI coding agents are productive precisely because they collapse the boundary between human intent and shell execution. That collapse is incompatible with the way secrets are stored, surfaced, and consumed in modern development environments. Shroud restores the boundary: agents are demoted from holding secrets to *referring to them*, and the human operator's authority becomes the explicit, attenuable, auditable substrate over which the agent acts. Built on UCAN and DID, Shroud is a small intervention at the terminal layer that materially shrinks the prompt-exposure attack surface — without changing the loop the agent already loves.

References

[1] Anthropic. *Claude Code: an agentic command-line tool for software engineering.* 2025.

[2] OpenAI. *Codex CLI: an open-source coding agent.* 2025.

[3] Cursor. *Agent mode: autonomous coding sessions.* 2025.

[4] Gauthier, P. *Aider: AI pair programming in your terminal.* 2024.

[5] HashiCorp. *Vault: secrets management.* 2015–.

[6] AgileBits. *1Password CLI.* 2019–.

[7] Miller, M. *Robust Composition: Towards a Unified Approach to Access Control and Concurrency Control.* PhD thesis, JHU, 2006.

[8] W3C. *Decentralized Identifiers (DIDs) v1.0.* W3C Recommendation, 2022.

[9] Fission Codes. *UCAN: User-Controlled Authorization Networks.* Specification v0.10, 2024.

[10] W3C CCG. *The did:key method.* Draft, 2023.

[11] Genode Labs. *Genode operating system framework*.

[12] Watson, R., Anderson, J., Laurie, B., Kennaway, K. *Capsicum: practical capabilities for UNIX*. USENIX Security, 2010.

[13] Fission Codes. *Capability-based identity for the web.* 2024.

Appendix A — Reference Wire Format

Resolution request (CBOR, over Unix socket):

```
{
  "v": 1,
  "type": "resolve",
  "name": "OPENAI_KEY",
  "ucan": "eyJhbGc...",
  "cmd_hash": h'1f23...e9'
}
```

Response:

```
{ "v": 1, "ok": true, "value": "sk-..." }
```

or

```
{ "v": 1, "ok": false, "error": "ucan_expired" }
```

Appendix B — Journal Record

```
[2026-05-24T17:42:13.118Z]
  t: 1748112133118
  agent: did:key:z6MkfRandomAgentABC
  ucan: bafyreig...
  cap: secret/read vault://OPENAI_KEY
  cmd_hash: 1f23...e9
  prev: bafyreif... (chain link)
  this: bafyreih...
```

Appendix C — Operator Workflow

```
$ shroud init
  • generated did:key:z6MkOperator...
  • vault created at ~/.shroud/vault.age
```

```
  • passphrase set

$ shroud put OPENAI_KEY
  paste value, ^D when done:
  ********
  • stored: OPENAI_KEY

$ claude-code   # or codex, cursor agent, etc.
  • shroud-shim attached
  • agent DID: did:key:z6MkAgent...
  • requesting capability: secret/read vault://OPENAI_KEY
  [approve / deny] approve, valid 1h
  • UCAN issued bafyreif...

[agent emits]:
  curl -H "Authorization: Bearer $SHROUD_REF(OPENAI_KEY)" \
       https://api.openai.com/v1/models

[runtime resolves at exec; agent never sees the bearer]

$ shroud audit --since 1h
  3 resolutions, 0 denials, 0 alarms
```

Draft for community review. Comments to team@useshroud.com or @useshroud on X.