

RFC-006: Vault System Architecture

Status: Implemented **Date:** January 2026 **Author:** Derrell Piper ddp@eludom.net **Implementation:** vault.scm (887 lines)

Abstract

This RFC specifies the Vault system for the Library of Cyberspace: cryptographically sealed version control with SPKI authorization, progressive metadata, archival support, and integrated audit trails.

Motivation

Git is powerful but lacks:

1. **Cryptographic sealing** – GPG signing is optional and awkward
2. **Authorization model** – Anyone with access can commit
3. **Archival features** – No first-class backup/restore
4. **Audit integration** – History is mutable

The Vault wraps Git with:

- **seal-** commands that cryptographically sign operations
- **SPKI certificates** for authorization
- **Three archive formats** for different use cases
- **Integrated audit trail** for non-repudiation

CYBERSPACE VAULT

| | |
|--------------|----------------|
| seal-commit | seal-release |
| seal-archive | seal-publish |
| seal-verify | seal-subscribe |

Audit Trail

SPKI Certs

Git

Core Operations

seal-commit

Stage and commit changes in one operation.

```
(seal-commit message  
  #!key files catalog subjects keywords description preserve)
```

Parameters: - message - Commit message (required) - files - Specific files to stage (optional) - catalog - Enable catalog metadata - subjects - Subject headings - keywords - Search keywords - description - Extended description - preserve - Enable preservation metadata

Process: 1. Stage specified files (or all modified) 2. Create git commit 3. Save metadata (if catalog or preserve) 4. Record in audit trail (if signing key configured)

Example:

```
(seal-commit "Add authentication module"  
  files: ('auth.scm" "auth-test.scm")  
  catalog: #t  
  subjects: '("security" "authentication")  
  keywords: '("login" "oauth"))
```

seal-update

Pull latest changes from remote.

```
(seal-update #!key branch)
```

Like svn update - fetches and fast-forwards.

seal-undo

Undo changes.

```
(seal-undo #!key file hard)
```

- file - Restore specific file
- hard - Discard all uncommitted changes

seal-history

Show commit history.

```
(seal-history #!key count)
```

Displays decorated graph log.

seal-branch / seal-merge

Branch and merge operations.

```
(seal-branch "feature-auth" #!key from)
(seal-merge "feature-auth" #!key strategy)
```

Version Management

seal-release

Create cryptographically sealed release.

```
(seal-release version #!key message migrate-from)
```

Parameters: - version - Semantic version (X.Y.Z required)
- message - Release notes - migrate-from - Previous version
for migration tracking

Process: 1. Validate semantic version format 2. Get
current commit hash 3. Create annotated git tag 4. Sign
with SPKI (if configured) 5. Create migration marker (if
migrate-from specified)

Signature Storage:

```
;; .vault/releases/1.0.0.sig
(signature
  (version "1.0.0")
  (hash "abc123...")
  (manifest "(release \"1.0.0\" \"abc123\" 1767685100)")
  (signature #${ed25519-signature}))
```

seal-verify

Verify release signature.

```
(seal-verify version #!key verify-key)
```

Process: 1. Load signature file 2. Recompute manifest
hash 3. Verify Ed25519 signature

Archival System

seal-archive

Create sealed archive of a version.

```
(seal-archive version #!key format output)
```

Formats:

Tarball (default)

```
(seal-archive "1.0.0" format: 'tarball)
```

- Standard gzipped tarball
- No history included
- Smallest size

Git Bundle

```
(seal-archive "1.0.0" format: 'bundle)
```

- Full git history
- Can clone directly
- Medium size

Cryptographic (legacy)

```
(seal-archive "1.0.0" format: 'cryptographic)
```

- Tarball + SHA-512 hash + Ed25519 signature
- Tamper-evident
- Manifest for verification

Zstd+Age (preferred)

```
(seal-archive "1.0.0" format: 'zstd-age)
```

- Zstd compression (faster, better ratio than gzip)
- Age encryption (X25519/Ed25519 compatible)
- SHA-512 hash + Ed25519 signature
- Encrypted at rest
- See RFC-018: Sealed Archive Format for full specification

Cryptographic Archive Structure:

```
vault-1.0.0.archive      # Manifest  
vault-1.0.0.archive.tar.gz # Tarball (cryptographic)
```

Zstd+Age Archive Structure:

```
vault-1.0.0.archive      # Manifest  
vault-1.0.0.archive.tar.zst.age # Encrypted archive
```

Manifest (cryptographic):

```
(sealed-archive
  (version "1.0.0")
  (format cryptographic)
  (tarball "vault-1.0.0.archive.tar.gz")
  (hash "sha512:...")
  (signature "ed25519:..."))
```

Manifest (zstd-age):

```
(sealed-archive
  (version "1.0.0")
  (format zstd-age)
  (archive "vault-1.0.0.archive.tar.zst.age")
  (compression zstd)
  (encryption age)
  (recipients ("age1..."))
  (hash "sha512:...")
  (signature "ed25519:..."))
```

seal-restore

Restore from sealed archive.

```
(seal-restore archive #!key verify-key target identity)
```

Parameters: – verify-key – SPKI public key for signature verification – target – Extraction directory – identity – Age identity file for decryption (zstd-age format)

Process: 1. Read manifest 2. Verify hash (archive integrity) 3. Verify signature (if key provided) 4. Decrypt (zstd-age only, requires identity) 5. Extract to target directory

Replication Layer

See RFC-001 for complete specification.

seal-publish

Publish release to remote.

```
(seal-publish version #!key remote archive-format message)
```

Supports: – Git remotes (push tags) – HTTP endpoints (POST)
– Filesystem paths (copy)

seal-subscribe

Subscribe to releases from remote.

(seal-subscribe remote #!key target verify-key)

Downloads and optionally verifies remote releases.

seal-synchronize

Bidirectional sync.

(seal-synchronize remote #!key direction verify-key)

Configuration

vault-init

Initialize vault for repository.

(vault-init #!key signing-key)

Sets up: – Signing key configuration – Audit trail directory – Metadata directory

vault-config

Get/set configuration.

(vault-config 'signing-key) ; Get

(vault-config 'signing-key some-key) ; Set

Configuration Options:

| Key | Type | Description |
|----------------|--------|--|
| signing-key | blob | Ed25519 private key for signing |
| verify-key | string | Path to verification public key |
| archive-format | symbol | Default: tarball, bundle, cryptographic, or zstd-age |
| age-recipients | list | Age public keys for encryption (zstd-age format) |
| age-identity | string | Path to age identity file for decryption |
| migration-dir | string | Directory for migration scripts |

| Key | Type | Description |
|----------------|---------|-----------------------------|
| track-metadata | boolean | Auto-stage metadata files |
| publish-remote | string | Default publication target |
| subscribe-dir | string | Directory for subscriptions |

Directory Structure

```

project/
  .vault/
    metadata/          # Commit metadata files
      abc123.sex
      def456.sex
    releases/         # Release signatures
      1.0.0.sig
      1.1.0.sig
    audit/            # Audit trail
      1.sex
      2.sex
    subscriptions/    # Downloaded releases
      vault-1.0.0.archive
  migrations/        # Version migration scripts
    1.0.0-to-2.0.0.scm
  .git/              # Git repository

```

Migration Support

Creating Migrations

```
(seal-release "2.0.0" migrate-from: "1.0.0")
```

Generates template:

```
;; migrations/1.0.0-to-2.0.0.scm
;;; Migration: 1.0.0 -> 2.0.0
;;; Generated: 1767685100
```

```
(define (migrate-1.0.0-to-2.0.0)
  ;; Define migration logic here
  #t)
```

```
(migrate-1.0.0-to-2.0.0)
```

Running Migrations

```
(seal-migrate "1.0.0" "2.0.0" #!key script dry-run)
```

Integrity Checking

seal-check

Verify vault integrity.

```
(seal-check #!key deep)
```

Checks: – Git repository health (git fsck) – Release signature validity (if deep) – Audit trail chain (if deep)

Security Model

Signing Key Handling

```
; Key is 64-byte Ed25519 secret key  
; First 32 bytes: seed  
; Last 32 bytes: public key
```

```
(define (get-vault-principal signing-key)  
  "Extract public key from signing key"  
  (blob-copy signing-key 32 32))
```

Authorization Flow

1. **Configure:** (vault-init signing-key: key)
2. **Operate:** (seal-commit ...) signs with configured key
3. **Audit:** Entry includes actor's public key
4. **Verify:** (seal-verify ...) checks signature

Threat Mitigations

| Threat | Mitigation |
|----------------------|-----------------------------|
| Unauthorized commits | SPKI certificates required |
| Release tampering | Ed25519 signatures |
| History rewriting | Audit trail non-repudiation |
| Archive corruption | SHA-512 hash verification |

Integration Points

With Audit Trail (RFC-003)

All vault operations create audit entries:

```
(audit-append
  actor: (get-vault-principal signing-key)
  action: '(seal-commit "hash123")
  motivation: message)
```

With SPKI (RFC-004)

Signing keys are SPKI principals:

```
(make-key-principal (get-vault-principal signing-key))
```

With Metadata (RFC-005)

Progressive metadata via seal-commit parameters:

```
(seal-commit "msg" preserve: #t) ; Full preservation
```

Usage Examples

Basic Workflow

```
; Initialize
(vault-init signing-key: my-key)

; Daily work
(seal-commit "Add feature")
(seal-commit "Fix bug")

; Release
(seal-release "1.0.0" message: "Initial release")

; Archive
(seal-archive "1.0.0" format: 'cryptographic)

; Verify
(seal-verify "1.0.0" verify-key: "my.public")
```

Federation Workflow

```
;; Publisher
(seal-release "1.0.0")
(seal-publish "1.0.0" remote: "/shared/releases")

;; Subscriber
(seal-subscribe "/shared/releases" verify-key: publisher-pub)

;; Bidirectional
(seal-synchronize peer-remote direction: 'both)
```

Document Formats

The Vault preserves documents in canonical formats:

| Format | Extension | Purpose |
|------------|-----------|---|
| Markdown | .md | Source, editing, version control |
| HTML | .html | Web viewing, rich rendering |
| PDF | .pdf | Archival, printing, distribution |
| Plain Text | .txt | Universal compatibility, IETF tradition |

All formats are first-class citizens in the Vault. RFCs and declarations SHOULD be published in all four formats for maximum preservation and accessibility.

References

1. Git Internals – Plumbing and Porcelain
 2. RFC-001: Replication Layer
 3. RFC-003: Cryptographic Audit Trail
 4. RFC-004: SPKI Authorization
 5. RFC-005: Progressive Metadata Levels
 6. RFC-018: Sealed Archive Format
 7. Semantic Versioning 2.0.0
-

Changelog

- 2026-01-06 – Initial specification

Implementation Status: Complete **Test Status:** Passing
(test-vault-simple.scm, test-vault-metadata.scm) **Lines of**
Code: 887