

# RFC-001: Replication Layer for Library of Cyberspace

**Status:** Implemented **Date:** January 2026 **Author:** Derrell Piper ddp@eludom.net **Implementation:** vault.scm (lines 507-713)

---

## Abstract

This RFC specifies a replication layer for the Library of Cyberspace preservation architecture, enabling cryptographically sealed releases to be published, subscribed to, and synchronized across distributed locations while maintaining tamper-evident audit trails.

## Motivation

The Library of Cyberspace requires a distribution mechanism that:

1. **Preserves cryptographic authenticity** – Signatures travel with artifacts
2. **Enables offline verification** – No centralized authority required
3. **Records provenance** – All publication events are audited
4. **Supports multiple transports** – Git, HTTP, filesystem
5. **Maintains loose coupling** – Works for confederations of friends

Traditional package managers and distribution systems assume centralized registries and online verification. This replication layer is designed for decentralized, long-term preservation where trust is established through SPKI certificates and cryptographic seals.

## Design Principles

1. **Sealed Releases** – Only cryptographically signed releases can be published
2. **Transport Agnostic** – Same API works for git, HTTP, filesystem
3. **Audit Everything** – All replication events are recorded in tamper-evident log

4. **Verify Before Trust** – Subscribers must verify cryptographic seals
5. **Explicit Authorization** – SPKI certificates determine who can publish

## Specification

### Three Core Operations

1. **seal-publish** Publish a sealed release to a remote location.

```
(seal-publish version
  remote: target
  archive-format: format
  message: notes)
```

**Parameters:** – **version** – Semantic version string (e.g., “1.0.0”) – **remote** – Publication target (git remote, URL, or directory path) – **archive-format** – ‘tarball, ‘bundle, or ‘cryptographic (default) – **message** – Release notes (optional)

**Behavior:** 1. Verify release exists (creates if needed via seal-release) 2. Create cryptographic archive with: – Tarball of repository at version tag – SHA-512 hash of tarball – Ed25519 signature of hash – Manifest with version, hash, signature 3. Publish to remote based on type: – **Git remote:** Push tag, optionally upload archive – **HTTP URL:** POST archive to endpoint – **Filesystem:** Copy archive to directory 4. Record publication in audit trail with: – Actor (public key from signing key) – Action (seal-publish version remote) – Motivation (release notes) – Cryptographic seal (signature)

### Audit Entry Format:

```
(audit-entry
  (id "sha512:...")
  (timestamp "Mon Jan 5 23:38:20 2026")
  (sequence 1)
  (actor
    (principal #${public-key-blob})
    (authorization-chain))
  (action
    (verb seal-publish)
    (object "1.0.0")
    (parameters "/path/to/remote")))
```

```
(context
  (motivation "Published to filesystem")
  (language "en"))
(environment
  (platform "unknown")
  (timestamp 1767685100))
(seal
  (algorithm "ed25519-sha512")
  (content-hash "...")
  (signature "..."))
```

**2. seal-subscribe** Subscribe to sealed releases from a remote source.

```
(seal-subscribe remote
  target: local-path
  verify-key: public-key)
```

**Parameters:** – remote – Source location (git remote, URL, or directory) – target – Local path for downloaded archives (optional) – verify-key – Public key for signature verification (optional)

**Behavior:** 1. Discover available releases from remote: – **Git remote:** List tags – **HTTP URL:** GET /releases endpoint – **Filesystem:** List .archive files 2. Download cryptographic archives 3. Verify each archive: – Check manifest structure – Verify SHA-512 hash of tarball – Verify Ed25519 signature (if verify-key provided) 4. Extract verified archives to target directory 5. Record subscription in audit trail: – Count of releases downloaded – Source location – Verification status

**Security Consideration:** Without verify-key, subscription downloads archives but cannot verify authenticity. SPKI certificate chains should be used to establish trust.

**3. seal-synchronize** Bidirectional synchronization of sealed releases.

```
(seal-synchronize remote
  direction: 'both
  verify-key: public-key)
```

**Parameters:** – remote – Sync target (git remote, URL, or directory) – direction – 'both (default), 'push-only, or 'pull-only – verify-key – Public key for signature verification (optional)

**Behavior:** 1. Discover local and remote releases 2. Compare versions to determine: – Releases to push (local but not remote) – Releases to pull (remote but not local) 3. Execute publication for new local releases 4. Execute subscription for new remote releases 5. Record synchronization in audit trail: – Count pushed and pulled – Remote location – Direction

**Use Case:** Periodic sync between trusted peers in a confederation.

## Archive Format

### Cryptographic Archive Structure

```
vault-1.0.0.archive          # Manifest file
vault-1.0.0.archive.tar.gz   # Actual tarball
```

### Manifest S-expression:

```
(sealed-archive
  (version "1.0.0")
  (format cryptographic)
  (tarball "vault-1.0.0.archive.tar.gz")
  (hash "sha512:...")
  (signature "ed25519:...")
  (timestamp 1767685100)
  (sealer #${public-key-blob}))
```

**Verification Steps:** 1. Read manifest 2. Hash tarball with SHA-512 3. Verify hash matches manifest 4. Verify Ed25519 signature on hash 5. Check SPKI authorization (optional)

## Transport Implementations

### Git Remote

- Uses git push to share tags
- Optionally uploads archives as release assets (GitHub, GitLab)
- Fetch uses git fetch + git tag -l

### HTTP Endpoint

- POST to /releases/ for publication
- GET /releases for discovery
- Content-Type: application/x-sealed-archive

## Filesystem

- Copy archives to shared directory
- Directory structure: <remote>/<archive-name>
- No network required, works with NFS, USB drives, etc.

## Audit Integration

Every replication operation creates an audit entry with:

1. **Content-addressed ID** – SHA-512 hash of entry
2. **Chained structure** – References parent entry
3. **SPKI principal** – Public key of actor
4. **Dual context** – Human motivation + machine environment
5. **Cryptographic seal** – Ed25519 signature

This provides: – **Non-repudiation** – Cannot deny publication  
– **Tamper evidence** – Changes are detectable – **Causality** –  
Chain shows temporal order – **Accountability** – Know who  
published what when

## Security Considerations

### Threat Model

**Trusted:** – Local filesystem and vault – SPKI private keys  
– Cryptographic primitives (libsodium)

**Untrusted:** – Remote repositories – Network transport –  
Downloaded archives – Remote publishers (until SPKI verified)

### Attack Scenarios

1. **Malicious Archive Substitution**
  - Attacker replaces archive on remote
  - **Mitigation:** Signature verification fails
2. **Version Rollback Attack**
  - Attacker removes newer releases
  - **Mitigation:** Audit trail shows previous versions
3. **Unauthorized Publication**
  - Attacker publishes fake release
  - **Mitigation:** SPKI authorization chain required
4. **Transport Tampering**
  - Network attacker modifies download
  - **Mitigation:** Hash and signature verification

## Best Practices

1. **Always verify signatures** – Use `verify-key` parameter
2. **Check SPKI certificates** – Verify authorization chain
3. **Maintain audit trail** – Detect suspicious patterns
4. **Use HTTPS for HTTP transport** – Prevent network attacks
5. **Backup signing keys** – Use Shamir secret sharing

## Implementation Notes

### Helper Functions

```
(tag-exists? tag-name)           ; Check if git tag exists
(git-remote? str)                ; Detect git remote format
(http-url? str)                  ; Detect HTTP/HTTPS URL
(publish-filesystem remote version archive) ; Copy to directory
(publish-http url version archive) ; POST to endpoint
```

### Dependencies

- **Git** – For version control and tag management
- **libsodium** – Ed25519 signatures, SHA-512 hashing
- **Chicken Scheme modules:**
  - (chicken process) – Run git commands
  - (chicken file) – Filesystem operations
  - (chicken irregex) – URL/remote detection

## Compatibility

This specification is compatible with:

- **Git tags** – Standard git operations
- **Git bundles** – Portable repository format
- **Tarball archives** – Universal archive format
- **S-expressions** – LISP/Scheme readable format
- **SPKI/SDSI** – Authorization certificates

Future extensions may add: – **IPFS transport** – Content-addressed distribution – **Tor hidden services** – Anonymous publication – **Encrypted archives** – Confidential distribution – **Multi-signature releases** – Threshold authorization

## Test Coverage

See `test-replication.scm`:

```
;; Test seal-publish to filesystem
(seal-publish "1.0.0"
  remote: "/tmp/cyberspace-publish-test"
  message: "Published to filesystem")

;; Verify archive exists
(file-exists? "/tmp/cyberspace-publish-test/vault-1.0.0.archive")

;; Verify audit entry created
(audit-read sequence: 1)
```

## References

1. **SPKI/SDSI** – RFC 2693, RFC 2692
2. **Content-Addressed Storage** – Git internals, IPFS
3. **Semantic Versioning** – semver.org
4. **Ed25519** – Bernstein et al.
5. **Audit Trails** – RFC-002 (Cryptographic Audit Trail)

## Changelog

- **2026-01-05** – Initial implementation and specification
  - seal-publish with git/HTTP/filesystem support
  - seal-subscribe with signature verification
  - seal-synchronize with bidirectional sync
  - Full audit trail integration
  - Cryptographic archive format

---

**Implementation Status:** ✓ Complete **Test Status:** ✓ Passing  
**Audit Integration:** ✓ Complete