RFC-010: Federation Protocol

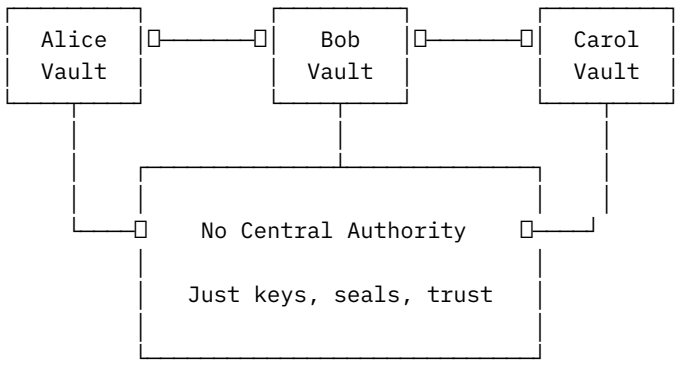
Status: Proposed **Date:** January 2026 **Author:** Derrell Piper ddp@eludom.net **Implementation:** Partial (see RFC-001 Replication Layer)

Abstract

This RFC specifies the Federation Protocol for the Library of Cyberspace: a peer-to-peer synchronization system enabling loose confederacies of friends to share and preserve cryptographically sealed artifacts without central authority.

E Pluribus Unum

Out of many, one.



Motivation

Centralized systems fail:

- **Single point of failure**: Server goes down, everyone stops
- **Censorship**: Authority can deny access
- **Trust concentration**: Must trust operator
- **Survival**: Company folds, data lost

Federation provides:

1. **Decentralized** - No master server
2. **Resilient** - Survives node failures
3. **Autonomous** - Each peer controls own data
4. **Cryptographic** - Trust through math, not authority
5. **Eventual consistency** - Convergence without coordination

---

Federation Model

Peer Relationships

```
Peer: A vault instance with identity (SPKI principal)

Relationships:
  - Publisher:  I push releases to you
  - Subscriber: I pull releases from you
  - Peer:       Bidirectional sync
```

Trust Model

```
(federation-trust
  (peer alice-pubkey
    (role publisher)
    (trust-level verified)     ; Signature verified
    (sync-policy automatic))

  (peer bob-pubkey
    (role subscriber)
    (trust-level known)        ; Key known, not verified
    (sync-policy manual)))
```

Trust levels: - **unknown**: Never seen, reject - **known**: Key registered, manual approval - **verified**: Signature chain verified via SPKI - **trusted**: Full automatic sync

---

Protocol Operations

Peer Discovery

```
(federation-discover)
;; Returns: List of known peers and their status
```

Discovery mechanisms: 1. **Explicit configuration**: Known peer list 2. **Git remotes**: Extract from repository 3. **Directory service**: Optional, not required 4. **mDNS/Bonjour**: Local network discovery

Peer Registration

```
(federation-register peer-uri
  #!key public-key trust-level)
```

Registers a new peer with: - URI (git remote, HTTP endpoint, filesystem path) - Public key for verification - Initial trust level

Release Announcement

```
(federation-announce version
  #!key peers message)
```

Pushes release notification to peers: 1. Create signed announcement 2. Send to specified peers (or all) 3. Peers verify signature 4. Peers decide whether to pull

Release Request

```
(federation-request version peer
  #!key verify-key)
```

Pulls specific release from peer: 1. Request release metadata 2. Verify signature 3. Download archive 4. Verify integrity 5. Record in audit trail

Synchronization

```
(federation-sync peer
  #!key direction verify-key)
```

Bidirectional sync (from RFC-001): 1. Exchange release lists 2. Identify missing releases 3. Push/pull as configured 4. Verify all signatures 5. Update audit trails

---

Message Format

Announcement Message

```
(federation-message
  (type announcement)
  (from #${alice-pubkey})
  (timestamp 1767685100)
  (payload
    (release "2.0.0")
    (hash "sha512:...")
    (archive-size 1048576)
    (notes "Major release"))
  (signature #${ed25519-sig}))
```

Request Message

```
(federation-message
  (type request)
  (from #${bob-pubkey})
  (timestamp 1767685200)
  (payload
    (release "2.0.0")
```

```
    (format cryptographic))
  (signature #${ed25519-sig}))
```

Response Message

```
(federation-message
  (type response)
  (from #${alice-pubkey})
  (in-reply-to "sha512:request-hash")
  (timestamp 1767685300)
  (payload
    (release "2.0.0")
    (archive-uri "/releases/vault-2.0.0.archive")
    (hash "sha512:...")
    (signature "ed25519:..."))
  (signature #${ed25519-sig}))
```

Transport Bindings

Git Transport

```
Origin: git@github.com:alice/vault.git
Mechanism: Tags + release assets

Announce: git push origin v2.0.0
Request:  git fetch origin --tags
Sync:     git fetch origin && git push origin
```

HTTP Transport

```
Endpoint: https://alice.example.com/federation

Announce: POST /federation/announce
Request:  GET /federation/releases/2.0.0
Sync:     POST /federation/sync
```

Filesystem Transport

```
Path: /shared/federation/alice

Announce: Copy to /shared/federation/alice/announce/
Request:  Read from /shared/federation/alice/releases/
Sync:     rsync --update
```

Conflict Resolution

Version Conflicts

Same version, different content:

```
(federation-conflict
  (version "2.0.0")
  (local-hash "sha512:abc...")
  (remote-hash "sha512:def...")
  (resolution reject))  ; Or: prefer-local, prefer-remote, rename
```

Default: Reject conflicts, require human decision.

Resolution Strategies

1. **Reject**: Stop sync, alert human
2. **Prefer-local**: Keep local version
3. **Prefer-remote**: Take remote version
4. **Rename**: Keep both as 2.0.0-local, 2.0.0-remote
5. **Merge**: If content mergeable (future)

Consistency Model

Eventual Consistency

- No global ordering required
- Each peer has local view
- Convergence through sync
- Conflicts resolved locally

Causal Ordering

Within a peer's releases: - Version numbers are monotonic - Audit trail provides causality - Hash chains prevent reordering

No Coordination

- No consensus protocol required
- No distributed lock
- No leader election
- Each peer autonomous

Security Considerations

Threat Model

Protected: - Unauthenticated release injection (signature verification) - Content tampering (hash verification) - Impersonation (SPKI principal binding) - Replay attacks (timestamps, sequence numbers)

Not protected: - Denial of service (rate limiting helps) - Privacy of release metadata (encrypted transport helps) - Sybil attacks (trust management helps)

Trust Verification

```
(define (verify-peer-message msg peer-key)
  (and (verify-signature msg peer-key)
       (verify-timestamp msg (current-seconds))
       (verify-not-replayed msg)))
```

Rate Limiting

```
(federation-config
  (rate-limit
    (announcements-per-hour 10)
    (requests-per-minute 60)
    (sync-interval-minimum 300)))
```

Gossip Protocol (Future)

For larger networks:

```
Alice announces to Bob and Carol
Bob announces to Dave and Eve
Eve announces to Frank

Result: Epidemic spread without central broadcast
```

Properties: - Logarithmic propagation time - Resilient to node failures - No single bottleneck

---

Bootstrap Procedure

New Peer Joining

1. Generate keypair
2. Register with known peer
3. Exchange public keys (out-of-band verification)

4. Initial sync to get current releases
5. Begin participating in federation

Network Partitions

- Partitions heal automatically on reconnection
- Conflicting releases detected and flagged
- Audit trails show partition history

Configuration

```
(federation-config
  ;; Identity
  (identity my-private-key)

  ;; Peers
  (peers
    (peer "alice" uri: "git@github.com:alice/vault.git"
                  key: alice-pubkey
                  trust: verified)
    (peer "bob"   uri: "/shared/bob-vault"
                  key: bob-pubkey
                  trust: known))

  ;; Policies
  (auto-sync #t)
  (sync-interval 3600)  ; seconds
  (verify-before-accept #t)

  ;; Security
  (require-signature #t)
  (trust-on-first-use #f))
```

Implementation Status

Implemented (RFC-001)

- seal-publish: Push to single remote
- seal-subscribe: Pull from single remote
- seal-synchronize: Bidirectional with single peer
- Transport: git, HTTP, filesystem

Proposed (This RFC)

- Multi-peer management
- Trust levels and policies
- Announcement protocol
- Gossip propagation
- Conflict resolution UI

References

1. Birman, K. (2007). The Promise, and Limitations, of Gossip Protocols.
2. Demers, A., et al. (1987). Epidemic Algorithms for Replicated Database Maintenance.
3. Shapiro, M., et al. (2011). Conflict-Free Replicated Data Types.
4. RFC-001: Replication Layer
5. RFC-004: SPKI Authorization

Changelog

- **2026-01-06** - Initial specification

Implementation Status: Partial (replication layer complete) **Protocol Status:** Proposed **Next Steps:** Multi-peer management, gossip propagation