RFC-004: SPKI Authorization Integration

Status: Implemented **Date:** January 2026 **Author:** Derrell Piper ddp@eludom.net
Implementation: cert.scm, spki-cert.scm, spki-verify.scm

Abstract

This RFC specifies the SPKI/SDSI certificate system for Cyberspace, providing authorization without identity. Principals are identified by cryptographic keys, not names. Authorization flows through verifiable delegation chains.

Motivation

X.509 certificates bind names to keys. This requires: - Certificate authorities (trust hierarchies) - Global name registries (DNS) - Identity verification (bureaucracy)

SPKI inverts this model:

> **Keys are principals. Authorization is local. Delegation is explicit.**

Benefits: - **No CA required** - Trust flows from keys you choose - **No global names** - Local namespaces, local meanings - **No identity** - Grant permissions to keys, not people - **Auditable** - S-expression format is human-readable

---

Specification

Principals

A principal is an authorization endpoint. Two types:

Key Principal Direct identification by public key:

```
(define-record-type <key-principal>
  (make-key-principal public-key)
  key-principal?
  (public-key principal-public-key))
```

S-expression: bare bytes

```
#${32-byte-ed25519-public-key}
```

Key Hash Principal Identification by hash of public key:

```
(define-record-type <keyhash-principal>
  (make-keyhash-principal hash-alg hash)
  keyhash-principal?
  (hash-alg principal-hash-alg)
  (hash principal-hash))
```

S-expression:

```
(hash sha512 #${64-byte-hash})
```

Authorization Tags

Tags define what permissions are granted:

```
(define-record-type <tag>
  (make-tag sexp)
  tag?
  (sexp tag-sexp))
```

Example tags:

```
;; Read access to library
(read (path /library/lamport-papers))

;; Agent spawning limit
(spawn-agent (max-count 5))

;; HTTP API access
(http-api (method POST) (path /deploy/*))

;; All permissions (wildcard)
(*)
```

Validity Period

Optional time constraints:

```
(define-record-type <validity>
  (make-validity not-before not-after)
  validity?
  (not-before validity-not-before)   ; ISO 8601 string
  (not-after validity-not-after))    ; ISO 8601 string
```

Certificate Structure

```
(define-record-type <cert>
  (make-cert issuer subject tag validity propagate)
  cert?
  (issuer cert-issuer)         ; Principal granting permission
  (subject cert-subject)       ; Principal receiving permission
  (tag cert-tag)               ; What is being granted
  (validity cert-validity)     ; When valid (optional)
  (propagate cert-propagate))  ; Can subject re-delegate?
```

S-expression format:

```
(cert
  (issuer #${alice-public-key})
  (subject #${bob-public-key})
  (tag (read (path /library/*)))
  (valid
    (not-before "2026-01-01")
    (not-after "2026-12-31"))
  (propagate))
```

Signed Certificate

```
(define-record-type <signed-cert>
  (make-signed-cert cert signature)
  signed-cert?
  (cert signed-cert-cert)
  (signature signed-cert-signature))

(define-record-type <signature>
  (make-signature hash-alg cert-hash sig-bytes)
  signature?
  (hash-alg signature-hash-alg)
  (cert-hash signature-cert-hash)
  (sig-bytes signature-sig-bytes))
```

Operations

Creating Certificates

```
(define cert
  (create-cert
    (make-key-principal alice-public)
    (make-key-principal bob-public)
    (make-tag '(read (path /library/*)))
    validity: (make-validity "2026-01-01" "2026-12-31")
    propagate: #t))
```

Signing Certificates

```
(define signed-cert
  (sign-cert cert alice-private))
```

Process: 1. Convert certificate to canonical S-expression 2. Hash with SHA-512 3. Sign hash with Ed25519 4. Create signature record 5. Combine into signed certificate

Verifying Certificates

```
(verify-signed-cert signed-cert alice-public)
```

Verification: 1. Recompute canonical S-expression 2. Hash with SHA-512 3. Compare with stored hash 4. Verify Ed25519 signature

Verifying Delegation Chains

```
(verify-chain root-key cert-list target-tag)
```

Chain verification ensures: 1. Each certificate is validly signed 2. Issuer of cert[n+1] matches subject of cert[n] 3. Tags are properly delegated (each implies the next) 4. Propagation is allowed (except final cert) 5. Final tag implies target tag

---

CLI Tools

spki-keygen

Generate Ed25519 keypair:

```
$ ./spki-keygen alice
Generated keypair:
  Public:  alice.public
  Private: alice.private
```

spki-cert

Create and sign certificate:

```
$ ./spki-cert \
    --issuer alice.private \
    --subject bob.public \
    --tag '(read (path /library/*))' \
    --propagate \
    --not-after "2026-12-31" \
    --output alice-to-bob.cert
```

spki-verify

Verify certificate signature:

```
$ ./spki-verify alice.public alice-to-bob.cert
✓ Certificate signature valid
```

spki-show

Display certificate in human-readable form:

```
$ ./spki-show alice-to-bob.cert
Certificate:
  Issuer:  ed25519:cbc9b260da65f6a7...
  Subject: ed25519:a5f8c9e3d2b1f0e4...
  Tag:     (read (path /library/*))
  Valid:   until 2026-12-31
  Propagate: yes
```

Tag Semantics

Tag Implication

Tag A implies Tag B if A grants at least all permissions of B.

```
(define (tag-implies tag1 tag2)
  (cond
    ((all-perms? tag1) #t)    ; (*) implies everything
    ((all-perms? tag2) #f)    ; Only (*) implies (*)
    (else (equal? tag1 tag2)))) ; Simple equality (extensible)
```

Standard Tag Vocabulary

Tag	Meaning
(*)	All permissions
(read (path P))	Read access to path P
(write (path P))	Write access to path P
(spawn-agent (max-count N))	Spawn up to N agents
(http-api (method M) (path P))	HTTP API access
(seal-release)	Permission to create releases
(seal-publish (remote R))	Permission to publish to R

Delegation Chains

Example: Three-Level Delegation

Alice (root) → Bob (admin) → Carol (operator)

Certificates:

```
;; Alice grants admin to Bob
(cert
  (issuer #${alice-key})
  (subject #${bob-key})
  (tag (*))
  (propagate))

;; Bob grants operator to Carol
(cert
  (issuer #${bob-key})
  (subject #${carol-key})
  (tag (seal-publish (remote origin))))
```

Verification:

```
(verify-chain alice-public
              (list alice-to-bob bob-to-carol)
              (make-tag '(seal-publish (remote origin))))
;; => #t if Carol can publish
```

Security Considerations

Threat Model

Trusted: - Local key storage - Ed25519/SHA-512 (libsodium) - Certificate chain construction

Untrusted: - Certificate sources - Network transport - Certificate claims (until verified)

Attack Mitigations

Attack	Mitigation
Certificate forgery	Ed25519 signatures
Unauthorized delegation	Propagate flag
Expired permissions	Validity period checks
Over-delegation	Tag implication checking

Key Management

- **Generation:** Use secure random (libsodium)
- **Storage:** Private keys in protected files
- **Backup:** Shamir secret sharing (see RFC-001)
- **Rotation:** Issue new certs, revoke old

Integration Points

Vault Authorization

```
(vault-init signing-key: alice-private)
(seal-release "1.0.0")  ; Requires seal-release tag
```

Audit Trail Attribution

```
(audit-append
  actor: bob-public
  action: '(seal-commit "abc123")
  authorization-chain: (list alice-to-bob-cert))
```

Replication Access Control

```
(seal-publish "1.0.0"
  remote: "origin"
  authorization: bob-to-carol-cert)
```

SPKI vs X.509

Aspect	X.509	SPKI
Identity	Names (DN)	Keys
Trust	CA hierarchy	Local choice
Namespaces	Global (DNS)	Local
Revocation	CRL/OCSP	Validity periods
Format	ASN.1/DER	S-expressions
Readability	Requires tools	Human-readable
Delegation	Implicit (CA)	Explicit (propagate)

References

1. Ellison, C., et al. (1999). SPKI Certificate Theory. RFC 2693.
2. Ellison, C., et al. (1999). SPKI Requirements. RFC 2692.
3. Rivest, R., & Lampson, B. (1996). SDSI - A Simple Distributed Security Infrastructure.
4. Lampson, B. (1971). Protection.

Changelog

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

Implementation Status: Complete **Test Status:** Passing (test-cert-minimal.scm)
CLI Tools: spki-keygen, spki-cert, spki-verify, spki-show