RFC-008: Shamir Secret Sharing

Status: Implemented **Date:** January 2026 **Author:** Derrell Piper ddp@eludom.net **Implementation:** crypto-ffi.scm (shamir-split, shamir-reconstruct)

Abstract

This RFC specifies Shamir's Secret Sharing implementation for the Library of Cyberspace, enabling K-of-N threshold splitting and reconstruction of cryptographic keys and other secrets.

Motivation

Private keys are single points of failure:

- **Key loss:** Funds locked forever
- **Key theft:** Complete compromise
- **Key escrow:** Trust a third party?

Shamir's Secret Sharing provides:

1. **Threshold recovery:** Any K of N shares reconstruct
2. **Information-theoretic security:** K-1 shares reveal nothing
3. **Distributed custody:** No single holder
4. **Backup flexibility:** Geographic distribution

From Adi Shamir's 1979 paper "How to Share a Secret":

> *Split a secret into N pieces such that any K pieces suffice to reconstruct, but K-1 pieces reveal absolutely nothing.*

Specification

Share Record

```
(define-record-type <shamir-share>
  (make-shamir-share id threshold x y)
  shamir-share?
  (id share-id)            ; Symbol: share-1, share-2, etc.
  (threshold share-threshold)  ; K value
  (x share-x)                 ; X-coordinate (1 to N)
  (y share-y))                ; Y-coordinate (blob, same size as secret)
```

Splitting a Secret

```
(shamir-split secret #!key (threshold 3) (total 5))
```

Parameters: - `secret` - Blob to split (any size, typically 32 or 64 bytes) - `threshold` - Minimum shares to reconstruct (K) - `total` - Total shares to create (N)

Returns: List of N shamir-share records

Algorithm: 1. For each byte of secret: - Generate K-1 random coefficients - Coefficient[0] = secret byte - Polynomial: $f(x) = a_0 + a_1x + a_2x^2 + … + a_{k-1}x^{k-1}$ 2. Evaluate polynomial at x = 1, 2, …, N 3. Package (x, f(x)) pairs as shares

Reconstructing a Secret

```
(shamir-reconstruct shares)
```

Parameters: - `shares` - List of at least K shamir-share records

Returns: Reconstructed secret blob

Algorithm: 1. Take first K shares 2. For each byte position: - Use Lagrange interpolation - Compute f(0) = secret byte 3. Assemble reconstructed secret

---

Galois Field Arithmetic

Operations performed in $GF(2^8)$ with irreducible polynomial $x^8 + x^4 + x^3 + x + 1$:

```
(define gf256-primitive #x11b)  ; x^8 + x^4 + x^3 + x + 1

(define (gf256-mul a b)
  "Multiply in GF(2^8)"
  ...)

(define (gf256-inv a)
  "Multiplicative inverse in GF(2^8)"
  ...)
```

```
(define (gf256-poly-eval coeffs x)
  "Evaluate polynomial at x using Horner's method"
  ...)
```

GF(2^8) ensures: - All operations stay within byte range - No overflow issues - Proper field properties (every non-zero element has inverse)

Usage Examples

Basic Secret Splitting

```
(import crypto-ffi)

(sodium-init)

;; Create a 32-byte secret
(define secret (make-blob 32))
;; ... fill with secret data ...

;; Split into 5 shares, threshold 3
(define shares (shamir-split secret threshold: 3 total: 5))

;; Distribute shares to custodians
(print "Created " (length shares) " shares")
(print "Threshold: " (share-threshold (car shares)))
```

Reconstruction from K Shares

```
;; Collect any 3 shares
(define collected (list share-1 share-3 share-5))

;; Reconstruct
(define reconstructed (shamir-reconstruct collected))

;; Verify
(if (equal? (blob->hex secret) (blob->hex reconstructed))
    (print "  Reconstruction successful!")
    (print "  Reconstruction failed"))
```

Ed25519 Key Backup

```
;; Generate keypair
(define keypair (ed25519-keypair))
(define public-key (car keypair))
(define private-key (cadr keypair))

;; Split private key (5-of-7 for production)
(define key-shares (shamir-split private-key threshold: 5 total: 7))

;; Later: reconstruct and verify
(define recovered-key (shamir-reconstruct (take key-shares 5)))

;; Test: sign with recovered key
(define message "Test message")
(define signature (ed25519-sign recovered-key message))
(define valid? (ed25519-verify public-key message signature))

(if valid?
    (print "  Recovered key produces valid signatures!")
    (print "  Key recovery failed"))
```

Security Properties

Information-Theoretic Security

With K-1 shares: - **No information** about secret is revealed - **Not computationally hard** - literally impossible - Even infinite compute power cannot break

This is because K-1 points determine infinitely many degree-(K-1) polynomials.

Share Independence

Each share is uniformly random: - Looks like random bytes - No correlation between shares - Safe to store on untrusted media

Threshold Guarantee

Exactly K shares required: - K shares: reconstruction succeeds - K-1 shares: no information - K+1 shares: still works (overdetermined)

Threshold Selection Guidelines

Use Case	Threshold	Total	Rationale
Personal backup	2-of-3	Simple recovery	
Team key	3-of-5	Majority required	
Organization root	5-of-7	Supermajority	
Hardware ceremony	7-of-11	High assurance	
Paranoid	11-of-15	Maximum distribution	

Considerations

- **Availability:** Higher K = harder to recover
- **Security:** Lower K = easier to collude
- **Geography:** Consider time zones for ceremonies
- **Succession:** What if custodians unavailable?

Share Distribution

Physical Security

```
Share 1: Safe deposit box (Bank A)
Share 2: Home safe
Share 3: Attorney's vault
Share 4: Trusted family member
Share 5: Offshore location
```

Digital Storage

```
Share 1: Hardware security module
Share 2: Air-gapped laptop
Share 3: Encrypted USB (passphrase protected)
Share 4: Paper printout (secure location)
Share 5: Tattoo (not recommended)
```

Geographic Distribution

- Different jurisdictions
- Different failure domains
- Different time zones (for ceremonies)

Verification Without Reconstruction

For periodic verification that shares are intact:

```
;; Each custodian verifies their share
(define (verify-share share expected-id expected-threshold)
  (and (eq? (share-id share) expected-id)
       (= (share-threshold share) expected-threshold)
       (= (blob-size (share-y share)) expected-length)))
```

Full reconstruction should be rare: - Key rotation ceremonies - Emergency recovery - Succession events

---

Integration with Threshold Signatures

Two complementary approaches:

Shamir for Key Backup (This RFC)

```
Private key → split → N shares
Recovery: K shares → reconstruct → use key
```

Multi-Signature for Governance (RFC-007)

```
N parties → N keys → N signatures
Verification: count valid   K
```

Use Shamir when: - Backing up existing keys - Emergency recovery scenarios - Single key must be reconstructable

Use Multi-Sig when: - Ongoing governance decisions - Need audit trail of who signed - Asynchronous authorization

---

Security Considerations

Threats Mitigated

Threat	Mitigation
Key loss	Any K shares recover
Single compromise	Need K colluding
Insider attack	Distribute to independent parties
Coercion	Geographic/jurisdictional diversity

Threats Remaining

Threat	Notes
K colluding parties	Fundamental limitation
Poor share storage	Operational security
Side channels during reconstruction	Use secure environments
Weak random generation	Use libsodium

Operational Security

1. **Generation:** Air-gapped machine, secure random
2. **Distribution:** Out-of-band verification
3. **Storage:** Encrypted, physically secure
4. **Reconstruction:** Secure room, witnesses
5. **Destruction:** Secure wipe after use

Implementation Notes

Dependencies

- `libsodium` - Secure random number generation
- `srfi-4` - u8vectors for byte manipulation

Performance

- Split: O(N × K × secret_length)
- Reconstruct: O(K^2 × secret_length)
- GF(2^8) operations: O(1) per byte

Limitations

- Secret size: Arbitrary (but typically ≤ 64 bytes)
- Share count: Practical limit ~255 (byte x-coordinates)
- Threshold: 2 ≤ K ≤ N

References

1. Shamir, A. (1979). How to share a secret. Communications of the ACM.
2. Blakley, G. R. (1979). Safeguarding cryptographic keys.

3. Beimel, A. (2011). Secret-Sharing Schemes: A Survey.
4. NIST SP 800-57. Recommendation for Key Management.

Changelog

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

Implementation Status: Complete **Test Status:** Passing (test-shamir.scm) **Field Arithmetic:** GF(2^8)