# Zsub Self-Sovereign Cryptographic Mesh: Overview

**Version 0.1.9**
**Authored by the Zsub Team**
**Published Dec 28, 2025**

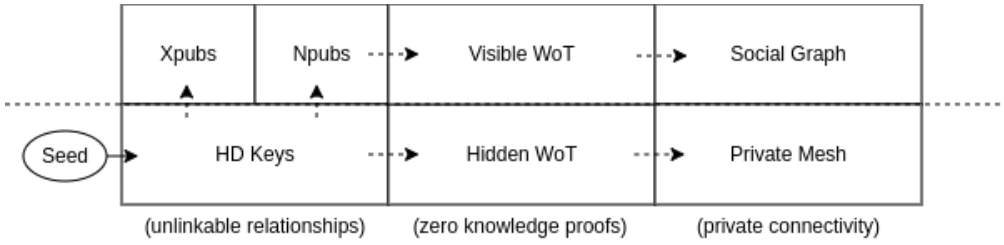
**Abstract:** We propose a protocol suite for seed-based self-custodial identity, hidden webs of trust, and secure private peer-to-peer connectivity without centralized intermediaries. The system composes protocols for hierarchical deterministic key management, privacy-preserving trust chains, secure UDP-based connections, onion routing, service provisioning, and economic incentives into a cryptographically unified architecture. Each component operates independently while combining to form an end-to-end infrastructure for securing digital autonomy.

## Motivation

Bitcoin delivers a pseudonymous, decentralized public monetary system without trusted intermediaries. Nostr provides a pseudonymous, decentralized public identity and social layer, enabling censorship-resistant social coordination. Yet pseudonymity without strong privacy is fragile: metadata, relationships, and behavioral patterns remain vulnerable to correlation, undermining the sovereignty these systems were built to protect.

Zsub closes this critical gap by design. It introduces true self-sovereign identity, privacy-preserving connectivity, and a hidden web of trust—unified at the cryptographic key layer and derived from a single seed. By shielding real individual identity, relationships, and network activity behind robust privacy guarantees, Zsub transforms the pseudonymity of systems like Bitcoin and Nostr into genuine, persistent autonomy. This seed-unified private foundation preserves the original promise of these protocols while enabling new classes of privacy-native network services.

Finally, privacy is an inseparable and necessary property of security. Even individuals and organizations that do not prioritize personal privacy—or might prefer a world of pervasive surveillance—face increasingly compelling security needs. As Bruce Schneier and others in the field have long argued, society faces a fundamental choice: surveillance capabilities that inevitably create systemic vulnerabilities, or genuine security through pervasive encryption. In an increasingly digital and multi-polar world, protocols like SSCM, by virtue of their *security properties*, should prove broadly applicable.



## Implementation Strategy

SSCM is achieved through step-wise delivery of its protocols across six phases. Each phase provides independently functional and valuable protocol implementations while building up a holistic infrastructure. This approach validates technical feasibility incrementally, continuously informs direction and allows deployment of useful subsets.

Initial implementation (Rebased, December 2025) demonstrates the viability of the secure runtime architecture, hierarchical key management and per-context isolation—seeding organic early adoption, dogfooding and real-world interoperation. See the roadmap for detailed phase specifications and development timeline.

To further ensure cryptographic correctness, implementation stability, and robustness against adversarial conditions, we employ fuzz testing on critical components and deterministic simulation testing for distributed system behavior.

## 1. System Architecture

SSCM comprises fifteen protocol components delivered over six development phases plus ongoing cross-cutting concerns:

### Identity and Trust (Phase 1)

1. **StarfortDB**: Hierarchical deterministic key database with continuous rotation, syncing and key exchange
2. **Heavy**: Privacy-preserving Web of Trust with minimal-disclosure proofs

### Connectivity (Phase 2)

3. **Keel**: UDP-based secure connections with double ratchet encryption and unlinkable mutual auth
4. **Submerge**: Message chunking and reassembly (transport)
5. **Rendezvous**: Blind peer coordination and message holding
6. **Skylight**: ICE-like NAT traversal via candidate exchange

### Integration (Phase 3)

7. **Surface**: Virtual layer 2 point-to-point with TAP interface
8. **Dock**: Service provisioning with cryptographic gating

### Scaling (Phase 4)

9. **Sonar**: Service discovery with reputation scoring
10. **Reactor Core**: Chaumian microcredits for private service access
11. **Fleet**: Incentivized network infrastructure

### Utilities (Phase 5)

12. **Coms**: Secure group messaging with attested membership

### Privacy (Phase 6)

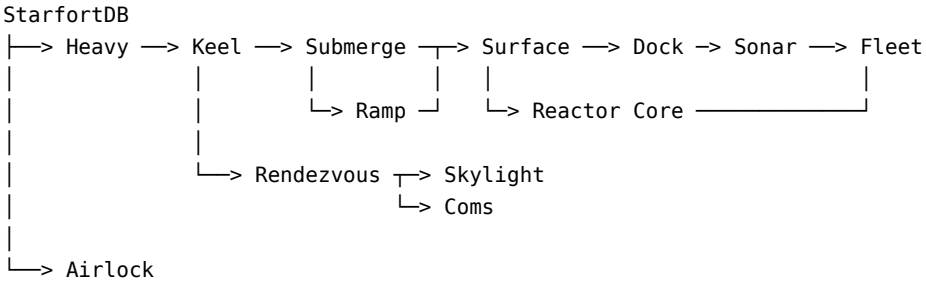
13. **Ramp**: Multi-hop onion routing
14. **Submerge+**: Path and session multiplexing

### Cross-Cutting

15. **Runtime**: Hull (sandboxing), Agent (key ops), Daemon (network), Helm (interface)
16. **Airlock**: Air-gapped key management on SeedSigners for high-security environments

### 1.1. Protocol Relationships

#### Dependency Graph




#### Layer Model

- **Layer 1 - Identity & Trust**: StarfortDB, Heavy
- **Layer 2 - Network & Transport**: Keel, Submerge, Rendezvous, Skylight
- **Layer 3 - Service**: Surface, Dock, Sonar
- **Layer 4 - Economics**: Reactor Core, Fleet
- **Layer 5 - Application**: Coms
- **Layer 6 - Network Privacy**: Ramp

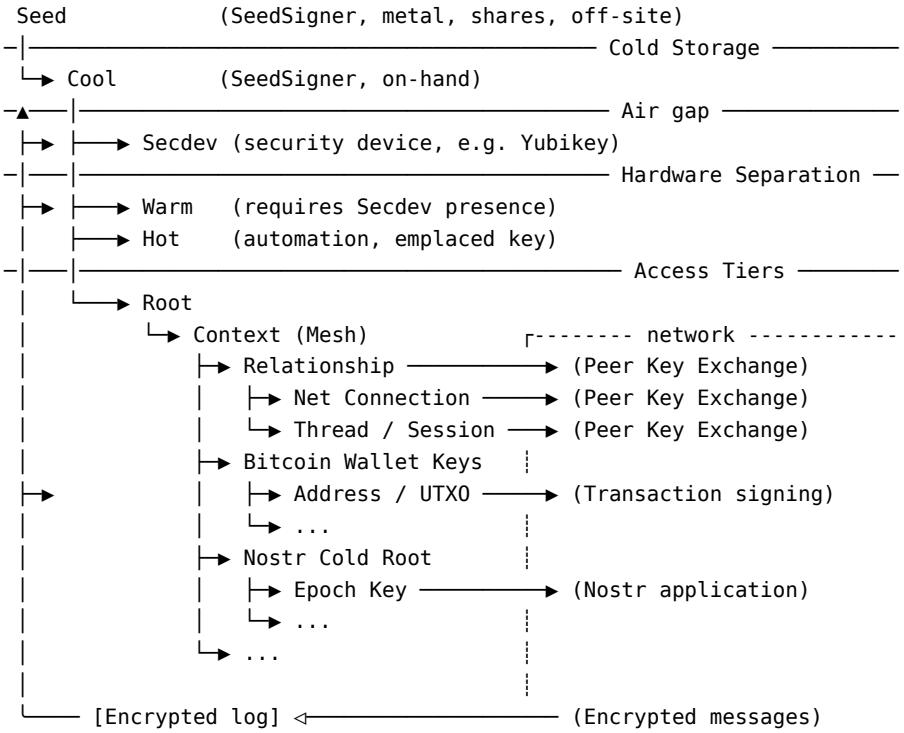
Higher layers depend on lower layers. Lower layers function independently.

### 1.2. Key Hierarchy and Access Model

#### Key Derivation, Access and Information Security Cycle

*Note:* This illustrates a **maximum-security** deployment model. Lower-security configurations are possible and supported (e.g., without security device or airgapping).

*Note:* Incoming encrypted messages / logs can target **any** key tier (cool/warm/hot), enabling use cases such as offline-only decryption for secure logs.



## 2. Technical Foundation

### 2.1. Cryptographic Primitives

**Core algorithms**:

- Ristretto255 over Curve25519 (all defaults)
- Ed25519 and Secp256 (where protocol compatibility requires)
- ChaCha20-Poly1305 (AEAD encryption)
- Blake3 (hashing and key derivation)
- Shamir secret shares

**Advanced protocols**:

- Double ratchet (Keel forward/backward secrecy)
- Schnorr proofs and Pederson Commitments (Heavy zero-knowledge authentication)
- MuSig2 and FROST (StarfortDB threshold signatures)
- Blind Schnorr signatures (Reactor Core payment privacy)

**Selection criteria**:

- Established specifications (RFCs where available)
- Suitability for resource-constrained devices
- Formal verification compatibility
- Clear post-quantum migration paths

### 2.2. Estimated Performance Characteristics

Approximate latencies on standard hardware (Intel i7, 16GB RAM):

| Operation | Latency |
|---|---|
| StarfortDB key derivation | ~0.1 ms |
| Heavy proof generation (5 hops) | ~0.7-1.2 ms |
| Heavy proof verification | ~0.8-1.3 ms |
| Keel handshake | ~5-15 ms |
| Ramp routing (3 hops) | ~15-30 ms |
| UC redemption | ~51 ms |
| Coms message encryption | ~0.1-0.2 ms |

See Performance Characteristics for detailed benchmarks.

### 2.3. Security Properties

**Privacy mechanisms**:

- Per-relationship key derivation (~$1/2^{256}$ unlinkability)
- Minimal-disclosure proofs (~$1/2^{128}$ collusion resistance for 5-hop chains)
- Onion routing with randomized paths
- Blind tokens decoupling purchase (Lightning) from service utilization
- Near-elimination of metadata exposure/linkability

**Security mechanisms**:

- Forward/backward secrecy via double ratchet
- Continuous key rotation
- Hardware security device support
- Process isolation (Hull sandboxing)
- Air-gapped operations (Airlock)

- Threshold recovery (M-of-N shares)

See Security and Threat Model for comprehensive analysis.

## 3. Protocol Implementations

### 3.0. Phase 0: Bootstrap Foundation (Completed December 2025)

**Deliverable**: Key management system with secure handling, full key isolation in agent, multi-tool integration and cli: bootstraps dogfooding and real-world user testing

- rebased git-based subcommand system, isolated agent (https://codeberg.org/zsub/rebased)
- Zig runtime (zero dependencies)
- Features: secure npub/nsec management, git/SSH/minisign compatible build signing
- Purpose: Validate core key derivation, per-context isolation, secure memory practices and runtime architecture, and enable early dogfooding/user testing in live workflows

**Status**: Implemented and actively used, remote signer coming soon.

### 3.1. Phase 1: Identity and Trust

**Deliverable**: Seed-derived private key database with multi-device sync and privacy-preserving trust proofs

**StarfortDB** provides hierarchical deterministic key management:

- Blake3-based key derivation with context strings
- Per-relationship isolation via HD keys
- Continuous key exchange protocol
- Multi-device synchronization with branch healing
- Tiered access control (cool/warm/hot keys)
- Continuous cascading rotation
- Hardware security device integration
- Threshold recovery via Shamir secret sharing

**Heavy** provides privacy-preserving trust verification:

- Schnorr proofs over Pedersen commitments
- Blake3 Merkle trees for revocation
- Configurable trust models (personal, organizational, federated)
- ~0.8-1.3 ms verification for 5-hop chains
- ~1 KB proof size

**Specification**: 1. Self-Sovereign Identity and Trust

### 3.2. Phase 2: Secure Connectivity

**Deliverable**: NAT-penetrating, double-ratchet encrypted P2P connections with blind peer rendezvous

**Keel** provides UDP-based secure connections:

- Double ratchet encryption (Blake3 symmetric, ECDH asymmetric)
- Heavy proof-based authentication
- Continuous key rotation (configurable intervals)
- Fast reconnection via proof tokens
- 473-byte payload limit in uniform 512-byte packets

**Skylight** provides NAT traversal:

- ICE-like candidate exchange (host, reflexive, relay)
- Authenticated connectivity checks
- Support for asynchronous operation

**Rendezvous** provides peer coordination:

- Blind message relaying
- Dead drop offline message holding
- Load balancing/distribution strategies

**Submerge** provides message handling:

- Chunking to transport MTU
- Reordering and reassembly
- Adaptive timeouts and congestion control

**Specification**: 2. Secure Connectivity

### 3.3. Phase 3: Application Integration

**Deliverable**: Virtual overlay networks + cryptographically gated service provisioning (TAP interface ready)

**Surface** provides virtual networking:

- Layer 2 virtual networks over Keel
- TAP interface for standard network stacks
- MAC/IP address derivation from StarfortDB keys
- OAuth2 adapter translating Heavy proofs to JWTs

**Dock** provides service provisioning:

- Service advertisement and session management
- Heavy proof-based access control
- Utility Credit integration

**Specification**: 3. Overlays and Services

### 3.4. Phase 4: Ecosystem Scaling

**Deliverable**: Decentralized service discovery, reputation, and incentivized node network with blind Lightning micro-payments

**Sonar** provides service discovery:

- Service advertisement via ServiceOffer
- Query-based discovery with filtering
- Quality-of-service metric integration
- Reputation scoring

**Reactor Core** provides payment infrastructure:

- Chaumian blind signatures (Ristretto255/Curve25519)
- 1,048,576 bucket hash ring for distribution
- Double-spending prevention via Seen sets
- Lightning Network integration

**Fleet** provides incentivized infrastructure:

- Third-party operated service nodes
- Lightning-based UC redemption settlement
- Quality-of-service reporting and verification

**Specification**: 4. Discovery and Incentives

### 3.5. Phase 5: Secure Messaging

**Deliverable**: Secure, forward-secret group messaging with attested membership (up to thousands of participants)

**Coms** provides group messaging:

- Ratchet-based encryption with forward secrecy
- Heavy-attested membership
- VRF-based cycle key agreement
- Support for 1,000-5,000 members

**Specification**: 5. Secure Messaging Utilities

### 3.6. Phase 6: Privacy Enhancements

**Deliverable**: Multi-hop onion routing + traffic obfuscation to defeat DPI and correlation attacks

**Ramp** provides onion routing:

- Multi-hop routing with layered encryption
- Randomized path selection
- Request tracks (sender-selected) and response tracks (random)
- 16-byte random Waypoint IDs

**Submerge+** provides advanced multiplexing:

- Path scattering across multiple channels
- Session multiplexing for traffic obfuscation
- SHA-256 PRNG for random assignment

**Specification**: 6. Privacy Enhancements

## 4. Implementation

### 4.1. Language and Dependencies

**Implementation language**: Zig

- Strong focus on simplicity, correctness, and explicitness (no hidden control flow or allocations)
- Zero third-party dependencies for maximum auditability and minimal attack surface
- Fast compilation enabling rapid iteration and quick security testing cycles
- Explicit memory management for thorough auditing and fine-grained control in security-critical code
- Seamless C interoperability for hardware security device integration (e.g., YubiKey/PC/SC)
- Efficient, small binaries ideal for embedded devices and resource-constrained environments

**Cryptographic implementations**:

- Zig standard library where available
- Custom implementations for protocol-specific requirements
- Formal verification via Tamarin where applicable

### 4.2. Platform Support

**Current**: Linux and MacOS (primary development)
**Planned**: iOS, Android, Windows

### 4.3. Runtime Architecture

**Hull**: Process isolation via Linux namespaces, seccomp filters, capability dropping
**Agent**: Handles sensitive cryptographic operations, enforces tiered key access
**Daemon**: Manages network I/O and lower-level protocol logic
**Helm**: Provides CLI and GUI interfaces for user interaction

**Specification**: A. Runtime Operations

### 4.4. Security Mechanisms

**Key protection**:

- Tiered access control (cool/warm/hot layers)
- Hardware security device support (YubiKey via PC/SC)
- Ephemeral in-memory decryption with mlock()
- Explicit zeroing on deallocation
- Threshold recovery (M-of-N Shamir shares)

**Process isolation**:

- Namespace isolation (PID, user, mount, network)
- Seccomp syscall filtering
- Capability dropping
- Ephemeral state in tmpfs

**Air-gapped operations**:

- Cool key derivation offline
- Security device provisioning
- Key rotation without network exposure

**Specification**: B. Secure Recovery and Custody

## 5. Development Status

**Completed**: rebased (December 2025)
**Current**: StarfortDB implementation (Q1 2026 target)
**Planned**: Complete SSCM protocol suite (Q2 2027 target)

### Timeline

- **Q1 2026**: Phase 1 (StarfortDB + Heavy)
- **Q2 2026**: Phase 2 (Keel + Skylight + Rendezvous + Submerge)
- **Q3 2026**: Phase 3 (Surface + Dock)
- **Q4 2026**: Phase 4 (Sonar + Reactor Core + Fleet)
- **Q1 2027**: Phase 5 (Coms)
- **Q2 2027**: Phase 6 (Ramp + Submerge+)

See Roadmap for detailed phase specifications and milestones.

## Series Index

### Main Series



### Supplementals