# HekateForge Shared Entropy Pool Choreography (SEPC)

## Version 2.1

By: **William Appleton**

Patent Pending, Application #3278624

Date Authored: August 20th 2025

# Abstract

Introducing the SEPC Protocol by HekateForge - A next-gen cryptographic framework designed for quantum-resilient, deniable communication across decentralized networks.

SEPC is a quantum-resilient, index-encoded AEAD transport with hybrid PQ key exchange that obfuscates wire image and resists replay/traffic analysis; designed for validator gossip, bridges, and oracle meshes at single-digit-ms latency. Built for adversarial coordination, deniability, and quantum resilience in decentralized systems.

**Key Highlights for Blockchain Builders:**

*No Mathematical connection between plaintext and transmission → radically reduces attack surface*

*Obfuscation using Entropy Pool mapping, ChaCha20Poly1305, AES-SIV, Kyber for secure key exchange*

*Deniability baked in → ideal for privacy-preserving DeFi, DAO ops, and zero-knowledge workflows*

*Quantum-safe design → future-proofing for post-quantum cryptography*

This protocol reimagines secure messaging for decentralized systems - where trustless coordination, anonymity, and resilience are non-negotiable.

# 1. Motivation

Standard encryption schemes rely on transmitting ciphertext, which - while undecipherable without keys - is still detectable, inspectable, and potentially vulnerable to future decryption. SEPC redefines this boundary by never transmitting the encrypted message content at all. Instead, it uses shared references into ephemeral entropy pools to encode and decode messages in an entirely choreography-based fashion, allowing for payloads to be calculated at the destination through the combination of the Pool, Index Transmission, and calculated HKDF keys required to decode the other two components.

# 2. Core Components

## 2.1 Entropy Pools

Entropy pools are several kilobyte segments of high-entropy random data with the following characteristics:

- Size: X secure shuffles of every possible byte value from 0 to 255. (30*256=7.5KB, 60*256=15KB)

- Entropy Requirement: Shannon entropy ≥ 7.5 bits per byte

- Distribution: all 256 byte values must be present in the pool

- Storage: AES-SIV encrypted and Base64-encoded for transport consistency

- Lifetime: 10-minute TTL with automatic destruction after single read (vault side)

**A decrypted pool would look like:**

```
{
"PoolID": "123456789",
"TTL": 1751729224,
"GeneratedAt": 1751727424,
"EnSrc": "V2.0 ",
"SHA256": "e3b0c44298fc1c149afbf4c8996fb924...",
"Data": "VGhpcyBpcyBhIDI1S0IgZW50cm9weSBjaHVuay4uLg=="
}
```

**Whereas an encrypted pool would look like:**


```
{
"Data": "AayBhIHVua1S0I y4HguayBhIy4uLIDI1L9IgZdfyBhI/fI1S0Igj=="
}
```


## 2.2 Pool Vault Infrastructure

The pool vault implements a secure ephemeral storage system:

- API: RESTful FastAPI interface

- Storage: FUSE filesystem with automatic file shredding

- Access Control: SHA-256 hashed pool identifiers

- Security: 3-pass overwrite deletion on read or expiry

- Validation: Entropy analysis and media detection

- Encryption: Vault sees AES-SIV encrypted blobs only, never keys or AEAD.

**Pool vault operations:**

- PUT /pool/{PoolIDHash}: Upload encrypted entropy pool

- GET /pool/{PoolIDHash}: Retrieve pool (single-use, auto-destruct)

- TIMEOUT: Any pool older than X minutes (default 10) is shredded

## 2.3 Key Exchange Layer

***Kyber key exchange:***

- Facilitates proven post-quantum shared secret exchange over insecure channel.

- shared secret is used with HKDF to derive keys for *AES-SIV key* for Pool encryption, *ChaCha20Poly1305 key* for the vpn data layer, *ChaCha20Poly1305 key* for the metadata handshake (Containing PoolID)

## 2.4 MetaData

***PoolID:***

- Integer range: 10,000 - 999,999,999

- Used to derive the PoolIDHash via SHA256 for encrypted pool retrieval from the Vault.

# 3. Protocol Operation

## 3.1 Message Encoding Process

**Step 1: Listening for Incoming Connection:**

Server mode peer opens a port and listens for connection requests. (default port 6969)

**Step 2: Kyber Key Exchange**

Sender and recipient exchanges metadata and keys via ChaCha20Poly1305 mode using the shared keys generated via HKDF and the secret derived from Kyber exchange over key port (default 6767).

**Step 3: Sender Entropy Pool Generation**

- Generate high-entropy pool data meeting requirements

- Create pool metadata with unique PoolID

- Encrypt pool with AES-SIV using pre-determined AES-SIV key

- Upload encrypted pool to vault

**Step 4: Sender Index Mapping**

- For each character c in Message_bytes:

- Find all positions where c appears in entropy pool

- Use cryptographically secure random selection of indexes for needed bytes

- Record selected position as index.

- Concatenate into 2-byte paired binary blob

- Result is PLD.

**Step 5: Transmission**

- Resulting PLD is wrapped in ChaCha20Poly1305 using the data layer key (predetermined) and transmitted to the receiving peer over data port (default 6969)

## 3.2 Message Decoding Process

**Step 1: Recipient Pool Retrieval**

- Request encrypted pool from vault using PoolIDHash

- Verify pool integrity and metadata

- Decrypt using AES-SIV with predetermined AES-SIV key.

**Step 2: Recipient Receives PLD**

- PLD is received from the sender peer.

- PLD is decrypted from ChaCha20Poly1305 using the predetermined key.

**Step 3: Recipient PLD Processing**

- Chunk binary PLD blob into 2 Byte indexes.

- Map each index to corresponding character in entropy pool

- Result is plaintext.

***VPN MODE* Step 4 (Re-key):**

- A new entropy pool of the same size is generated every X seconds (default 600)

- The new entropy pool is encoded with the old pool.

- The new entropy pool is prefixed with byte 255 (0xFF) and sent over the established tunnel.

- The other peer receives and decodes the tunnel packet prefixed with 255 (0xFF) and knows to swap the old pool with the new entropy.

## 4. Security Properties

**Transmission Unrelated to Plaintext:** No recoverable mathematical linkage between plaintext and on-the-wire representation under standard assumptions (AES-SIV is secure).

**Forensic Resistance:** The PLD transmission or entropy pool individually provide no information about the plaintext message due to random indexing across several instances of each Pool byte.

**Perfect Forward Secrecy:** Each entropy pool is destroyed after single use (File mode) or after TTL (VPN Mode) (Default 600 seconds), preventing retroactive decryption.

**Deniability:** Communications appear as random coordinate mappings with no provable connection to specific content.

**Quantum Readiness:** No reliance on difficult math to protect plaintext in transit.

## 5. Implementation Considerations

### 5.1 Performance Characteristics

- **Encoding Overhead:** ~2x message size due to 1:2 Byte mappings. (1 plaintext byte is 2 index bytes)

- **Pool Generation:** Computationally intensive entropy validation

### 5.2 Measured Performance (VPN MODE)

- *4-6ms pings over LAN tunnel* (360MBPS Wi-Fi LAN)

- *103ms pings over WAN tunnel* (Njella VPS in Oslo to DigitalOcean VPS in Toronto)

- *>3 hours proven tunnel life (python) with live ping over tunnel (600 second re-key)*

## 6. Applications

**Validator Gossip & Bridge Coordination** - SEPC's deniable, replay-resistant index streams allow validators and bridge operators to coordinate without exposing message content or timing patterns, ideal for MEV-sensitive environments.

**zkRollup Sequencer Messaging** - The protocol's ephemeral pool model and hybrid PQ handshake support sequencer-to-prover messaging with built-in forward secrecy and quantum resilience.

**Oracle Meshes & Off-Chain Data Sync** - SEPC enables secure, deniable transmission of off-chain data across oracle networks, with entropy pool TTLs enforcing freshness and preventing replay.

**DAO Ops & Governance Messaging** - SEPC's deniability and lack of reliance on mathematical hardness on the wire make it ideal for sensitive governance coordination, especially in adversarial or regulatory environments.

**Cross-Chain Messaging & Interop** - The protocol's index-encoded transport can be adapted for cross-chain message relays, with pool vaults acting as ephemeral message anchors.

## 7. Future Directions

1. **SEPC is engineered for integration into decentralized systems** requiring deniable, quantum-resilient messaging. A Rust-based implementation is underway to support validator gossip, oracle meshes, and bridge coordination at sub-10ms latency. The protocol's index-encoded AEAD transport and hybrid PQ key exchange make it ideal for rollup sequencers, zkVM coordination, and cross-chain messaging layers

2. **Future iterations** will explore integration with libp2p, QUIC-based wire images, and modular runtime environments like Cosmos SDK and Substrate. SEPC's ephemeral entropy pools and replay-resistant index streams offer a novel substrate for zero-knowledge messaging and privacy-preserving DAO coordination.