HekateForge Shared Entropy Pool Choreography (SEPC)

Version 2.0

By: **William Appleton**

Patent Pending, Application #3278624

Date Authored: August 15th 2025

Abstract

This whitepaper introduces the HekateForge Shared Entropy Pool Choreography (SEPC) Protocol - a novel communication framework that replaces conventional encrypted transmission with cryptographic referencing of shared entropy. No ciphertext version of the plaintext message ever traverses a network; instead, transmission occurs through ephemeral coordinate mappings into time-bound entropy spaces. By leveraging entropy pools, XOR-based obfuscation, ChaCha20Poly1305, AES-SIV, and secure key exchange through Elliptic Curve Diffie Hellman Curve25519, the protocol achieves deniable, and forensically resistant communication suitable for quantum-secure applications.

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 shuffles of every possible byte value from 0 to 255. (30 shuffles is 7.5KB 60 is 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 is shredded

2.3 Key Exchange Layer

X25519 Elliptic Curve Diffie-Hellman key exchange:

- Facilitates 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 and Key_1 for xor obfuscation of index mappings)

2.4 MetaData

Key_1 Value:

- Integer range: 0 to X (Where X is the maximum pool index IE 7680 for 7.5KB)

- Used in XOR index obfuscation.

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: X25519 DH Key Exchange

Sender and recipient exchanges metadata and keys via ChaCha20Poly1305 mode using the shared keys generated via HKDF and the secret derived from X25519 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.

Step 5: Sender Index Obfuscation

- Apply XOR chaining:

- Index_1: XOR with Key_1 (pre-determined)

- Index_n: XOR with previous decoded index

- Concatenate into 2-byte paired binary blob

- Result is PLD.

Step 6: 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.

- Apply reverse XOR chaining using same parameters (key_1)

Step 4: Recipient Message Recovery

- Map each index to corresponding character in entropy pool

- Result is plaintext.

***VPN MODE* Step 5:**

- 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 profixed 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

No Ciphertext in Transit: Only coordinate references are transmitted; the actual encrypted content never leaves the sender's system until decoded by the authorized recipient.

Forensic Resistance: The PLD transmission or entropy pool individually provide no information about the plaintext message.

Perfect Forward Secrecy: Each entropy pool is destroyed after single use, 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

- **Network Efficiency:** Single pool supports multiple message encodings before needing rotation

5.2 Measured Performance (VPN MODE)

- *5-6ms pings over LAN tunnel* (360MBPS Wi-Fi LAN)

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

6. Applications

Military/Government Communications: Deniable messaging with quantum-ready security for classified information exchange.

Medical Device Synchronization: HIPAA-compliant patient data coordination with forensic resistance.

SCADA System Coordination: Industrial control system communications resistant to traffic analysis.

Secure Corporate Messaging: Executive communications with built-in plausible deniability.

7. Future Directions

RUST Optimized implementation: The current implementation is limited due to being implemented in Python and in user-space. A C++ or Rust version would run much faster.