HekateForge Shared Entropy Pool Choreography (SEPC) Protocol Version 1.1

A Cryptographic Paradigm for Secure, Deniable, and Forensically Resistant Communication

By: **William Appleton**

Patent Pending, Application #3278624

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 ever traverses a network; instead, transmission occurs through ephemeral coordinate mappings into time-bound entropy spaces. By leveraging entropy pools, deterministic XOR-based segment encoding, and secure key exchange, the protocol achieves bandwidth-efficient, 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.

2. Core Components

2.1 Entropy Pools

Entropy pools are 25KB segments of high-entropy random data with the following characteristics:

- Size: Approximately 25,600 bytes of ASCII printable characters
- Character Set: 92 printable ASCII characters (!”#$%&’()*+,-./0-9:;<=>?@A-Z[]^_a-z{|}~)
- Entropy Requirement: Shannon entropy ≥ 7.5 bits per byte
- Distribution: All 92 characters must be present in the pool
- Storage: Base64-encoded for transport consistency
- Lifetime: 10-minute TTL with automatic destruction after single read

Entropy pools include metadata structured as JSON:

```
{
  "PoolID": "123456789",
  "TTL": 1751729224,
  "GeneratedAt": 1751727424,
  "EnSrc": "V1.1",
  "SHA256": "e3b0c44298fc1c149afbf4c8996fb924...",
  "Data": "VGhpcyBpcyBhIDI1S0IgZW50cm9weSBjaHVuay4uLg=="
}
```

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: AES-SIV with associated data binding

Pool vault operations:

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

2.3 Key Exchange Layer

Quantum Key Distribution (QKD) (Preferred):

- Transmits PoolID, LIC, segment length, and SEED securely
- Provides information-theoretic security

Public Key Infrastructure (PKI) (Classical fallback):

- Compatible with post-quantum algorithms
- Offers computational-level security
- Supports Elliptic Curve Diffie Hellman x448

2.4 Cryptographic Parameters

LIC Artifact (Password String):

- ASCII string used for key derivation

- Combined with SEED to generate XOR keys

SEED Value:

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

- Used in conjunction with LIC for key generation

Key Derivation for Cascading XOR Obfuscation:

Key_1 = SUM(LIC_bytes) × SEED

Segment Length:

- Configurable integer: 5-50 characters

- Determines post-XOR chaining block size

3. Protocol Operation

3.1 Message Encoding Process

Step 1: Message Preparation

Message_bytes = BASE64_ENCODE(message_file)

Step 2: Entropy Pool Generation

- Generate high-entropy ASCII data meeting requirements
- Create pool metadata with unique PoolID
- Encrypt pool with AES-SIV using SHA-512(LIC) as key
- Upload encrypted pool to vault

Step 3: Index Mapping

- For each character c in Message_bytes:
  - Find all positions where c appears in entropy pool
  - Use cryptographically secure random selection
  - Record selected position as index

Step 4: Index Encoding

- Concatenate indices using length-prefixed format:
  - Format: {length$_1$}{index$_1$}{length$_2$}{index$_2$}...
  - Example: “365825241973” represents indices 658, 52, 1973

Step 5: Segment Obfuscation

- Divide index string into fixed-length segments

- Apply XOR chaining:

  - Segment$_1$: XOR with Key$_1$

  - Segment$_n$: XOR with previous decoded segment

- Compress result with zlib level 9

- Output as PLD (Payload) file

3.2 Message Decoding Process

Step 1: Pool Retrieval

- Request encrypted pool from vault using PoolIDHash

- Verify pool integrity and metadata

- Decrypt using AES-SIV with SHA-512(LIC) key

Step 2: PLD Processing

- Decompress PLD file

- Apply reverse XOR chaining using same parameters

Step 3: Index Resolution

- Parse length-prefixed index format

- Map each index to corresponding character in entropy pool

- Reconstruct Base64 message string

Step 4: Message Recovery

- Decode Base64 to recover original message

- Output to specified file

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 file, metadata, and 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: Compatible with both QKD and post-quantum cryptographic primitives.

Traffic Analysis Resistance: Variable message lengths map to consistent entropy pool sizes, obscuring message size patterns.

5. Implementation Considerations

5.1 Performance Characteristics

- Encoding Overhead: ~3x message size due to Base64 and indexing

- Pool Generation: Computationally intensive entropy validation

- Network Efficiency: Single pool supports multiple message encodings

5.2 Operational Security

- Pool vault requires secure infrastructure with tamper-evident storage

- Key exchange must be performed over authenticated channels

- Entropy pool generation should use hardware random number generators

5.3 Error Handling

- Pool unavailability results in immediate decode failure

- Invalid parameters produce deterministic failure modes

- No partial message recovery to prevent information leakage

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

Decentralized Pool Vault Mesh: Distributed entropy storage across multiple nodes for increased availability and resistance to targeted attacks.

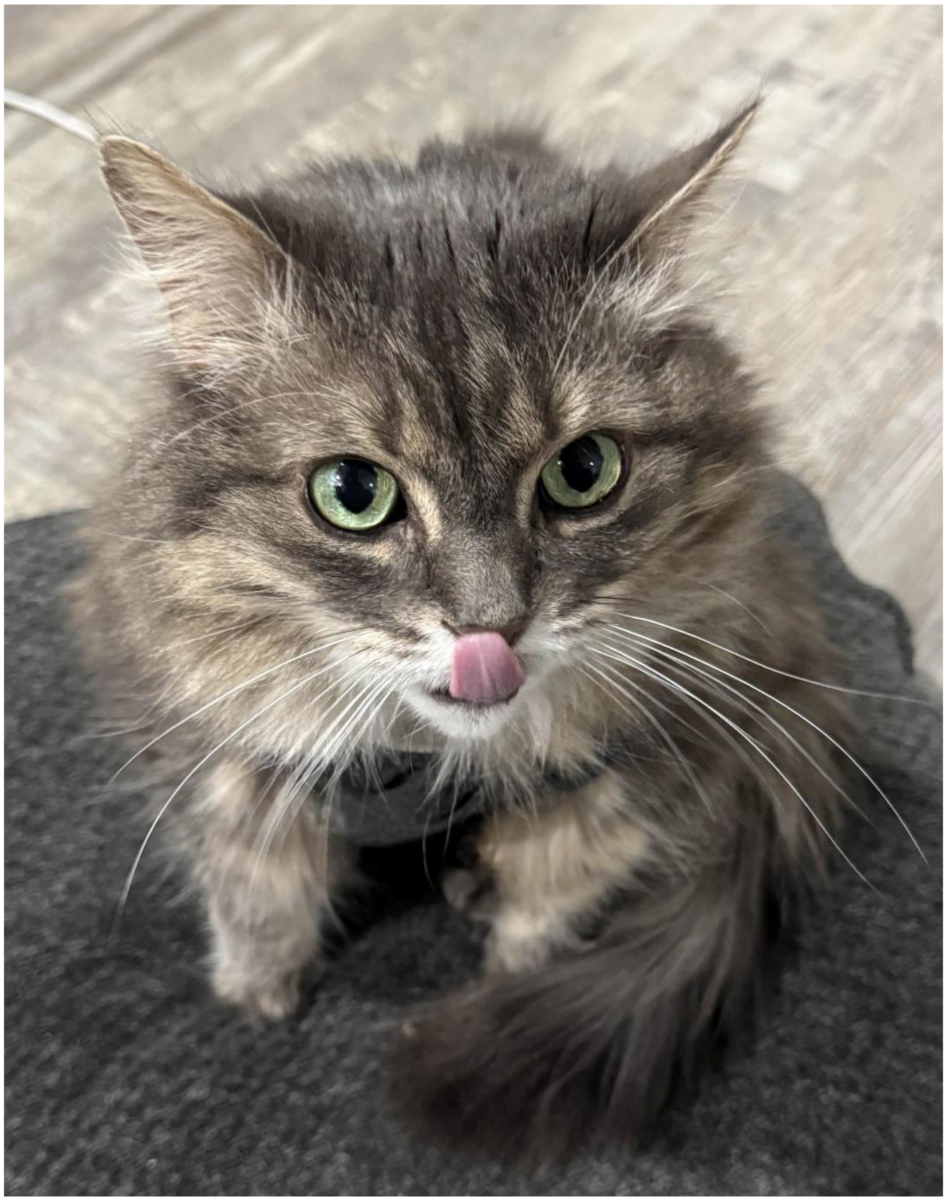
Bitstream Encoder Implementation: Direct binary encoding to eliminate Base64 overhead and improve efficiency.

Multi-party Group Chat: Choreography models supporting simultaneous multi-recipient messaging through shared entropy spaces.

Steganographic Pool Distribution: Embedding entropy pools within legitimate data streams to further obscure communication infrastructure.

8. Conclusion

The SEPC protocol represents a fundamental shift from traditional encryption paradigms by eliminating ciphertext transmission entirely. Through the innovative use of shared entropy spaces and coordinate-based messaging, SEPC achieves a unique combination of quantum-ready security, forensic resistance, and plausible deniability that addresses emerging threats in modern communication security.



HekateForge SEPC V1.1 - Dedicated to Jenga: 2014 - July 18th 2025