Independent Submission
Internet-Draft
Intended status: Experimental
Expires: 20 September 2026

S. Bellis
Unheaded

19 March 2026

Sophia Dictionary Format for the Unheaded Protocol
draft-bellis-unheaded-sophia-dictionary-00

Abstract

The Sophia Dictionary Format defines the serialization, storage, and distribution mechanism for semantic metadata that accompanies the Unheaded Protocol. Sophia dictionaries are exponent-decoding tables that translate compact byte values (0x00-0xFF) into meaningful human-readable categories (service identifiers, QoS classes, flow actions, etc.) and their associated metadata.

This memo specifies the CBOR serialization format for dictionary entries, the BPF map representation for in-kernel storage, the atomic update protocol for cluster-wide distribution via the Wotan memory bus, and the minimum required dictionary entries for any conformant Unheaded deployment.

Draft-03 introduces sub-dictionary type systems for hierarchical knowledge representation and QPACK compression headers for efficient dictionary entry encoding over the wire.

Sophia dictionaries support atomic replacement: updates propagate to all nodes in under 10 milliseconds without packet loss or service interruption.

Table of Contents

1. Introduction

1.1. Problem Statement

The Unheaded Protocol [UNHEADED-FOUNDATION] defines a 20-byte register file (the Monad) that travels with every packet. Each byte in the Monad is exponent-encoded: the actual value is reconstructed as base^exponent * multiplier. But where do the base, multiplier, and the semantic meaning of each byte position come from?

The answer is Sophia: a distributed, versioned dictionary system that maps byte values to meanings. Sophia is the semantic layer. Without it, the Monad fields carry no application semantics. With it, a 0x03 byte value resolves to "architect" or "realtime" or "forward" or "open" depending on the field position and active dictionary version.

This memo specifies:

1. How Sophia dictionaries are represented on the wire (CBOR format per RFC 8949)

2. How they are stored in BPF maps for nanosecond-latency lookups

3. How they are distributed to all nodes atomically via Wotan [WOTAN]

4. The minimum dictionary entries that all implementations MUST support

5. Version negotiation and backward-compatibility rules

6. Sub-dictionary type systems for hierarchical knowledge representation (NEW in draft-03)

7. QPACK compression headers for efficient dictionary encoding (NEW in draft-03)

1.2. Cross-References

This document is part of the Unheaded Protocol specification family:

* *Protocol Foundation* [UNHEADED-FOUNDATION]: Defines the Monad wire format (20 bytes, FROZEN at v0x01), per-hop processing, IANA registries, and the IANA registration procedures for new metric types.

* *Wotan Memory Protocol* [WOTAN]: Defines the memory and I/O bus including error code taxonomy, helper return codes, and error recovery procedures.

2. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 RFC2119 [RFC2119] RFC8174 [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. Terminology

Root Dictionary: The top-level BPF map (type BPF_MAP_TYPE_HASH) keyed by root entry ID (0x00-0xFF) that points to sub-dictionaries.

Sub-Dictionary: A BPF map (type BPF_MAP_TYPE_ARRAY_OF_MAPS) indexed by sub-entry ID (0x00-0xFF) that contains semantic metadata.

Nested Sub-Dictionary: A sub-dictionary that itself contains references to further sub- dictionaries, enabling hierarchical (tree-structured) knowledge representation. (NEW in draft-03)

Sophia Lookup: A two-level (or deeper for nested sub-dictionaries) hash table traversal: root_map[key0] -> sub_dict_id, then sub_dict[key0][key1] -> value.

Dictionary Version: An unsigned 8-bit counter (0-255) that increments with each dictionary update. Used for consistency validation across nodes.

Atomic Update: The act of replacing an entire BPF map and updating the array-of-maps reference in a single atomic kernel operation.

Wotan Topic: A publish-subscribe channel through which dictionary updates are broadcast. Format: sophia.dictionary.v{N} where N is the version number.

BPF (Berkeley Packet Filter): Per RFC 9669, the in-kernel virtual machine and map storage system. This memo uses "BPF" not "eBPF" per RFC 9669 conventions.

QPACK: Header compression format defined in RFC 9204, adapted in this specification for Sophia dictionary entry compression. (NEW in draft-03)

Anamnesis: The event sourcing subsystem that emits and logs system events (such as instruction traces, anomalies, and profiling metrics) for observability and debugging purposes.

4. Dictionary Model

4.1. Tree Structure

Sophia dictionaries are trees, not flat tables. The root level maps entry categories to sub-dictionaries. Each sub-dictionary maps specific values within that category to metadata.

Example:

```
Root entry 0x01 -> "service_identity" -> sub-dict #1
  Sub-dict #1[0x01] -> {name: "captain", ...}
  Sub-dict #1[0x02] -> {name: "timeguru", ...}
  Sub-dict #1[0x03] -> {name: "architect", ...}

Root entry 0x02 -> "flow_action" -> sub-dict #2
  Sub-dict #2[0x01] -> {name: "forward", ...}
  Sub-dict #2[0x02] -> {name: "trace", ...}
  Sub-dict #2[0x03] -> {name: "sample", ...}

The SAME byte 0x03 means:
  [0x01, 0x03] = service "architect"
  [0x02, 0x03] = action "sample"
  [0x03, 0x03] = qos "realtime"
```

This compositional structure provides 256^K total expressible meanings with K key positions, using only 2*K bytes on the wire (K bytes per lookup).

4.2. Sub-Dictionary Type System (NEW in draft-03)

4.2.1. Overview

Sub-dictionaries in draft-02 were flat: each sub-dictionary entry contained leaf metadata (name, endpoint, key material, etc.). Draft-03 introduces typed sub-dictionaries that MAY themselves contain references to nested sub-dictionaries, enabling hierarchical knowledge graphs.

4.2.2. Sub-Dictionary Types

Each sub-dictionary entry includes a type field that determines its structure:

Sub-Dictionary Type Codes:

0x00	LEAF	Leaf node: contains metadata only (draft-02 behavior)
0x01	BRANCH	Branch node: contains a reference to a nested sub-dictionary
0x02	COMPOSITE	Contains both metadata AND a nested sub-dictionary reference
0x03	ALIAS	Alias to another sub-dictionary entry (indirection)
0x04-0xFF		Reserved for future use

4.2.3. Nested Sub-Dictionary Structure

A BRANCH or COMPOSITE entry includes a nested_dict_id field:

```
struct sophia_typed_sub_entry {
    u8   entry_type;          // 0x00=LEAF, 0x01=BRANCH, 0x02=COMPOSITE,
                              // 0x03=ALIAS
    u8   name[32];            // Null-terminated name string
    u32  endpoint_ip;         // Service IPv6 last 32 bits (LEAF/COMPOSITE)
    u16  endpoint_port;       // Service port (LEAF/COMPOSITE)
    u8   pqc_algo;            // PQC algorithm ID (LEAF/COMPOSITE)
    u8   key_epoch;           // Key rotation counter (LEAF/COMPOSITE)
    u8   fingerprint[32];     // SHA3-256 of PQC public key (LEAF/COMPOSITE)
    u32  nested_dict_id;      // Nested sub-dict index (BRANCH/COMPOSITE)
    u16  reserved;
};  // Total: 84 bytes per entry
```

4.2.4. Lookup Chain for Nested Dictionaries

A nested Sophia lookup performs:

1. Look up root_key in sophia_root map
2. Extract sub_dict_id from the result
3. Look up sub_dict_id in sophia_dicts (array of maps)
4. Look up sub_key in the obtained sub-dictionary map
5. If entry_type == BRANCH or COMPOSITE:
   a. Extract nested_dict_id from the entry
   b. Look up nested_dict_id in sophia_dicts
   c. Look up next_key in the nested sub-dictionary
   d. Repeat until LEAF node is reached
6. Return the final LEAF value

Cost: Each additional nesting level adds one BPF hash lookup (~100ns). Total: ~300ns + 100ns per nesting level.

4.2.5. Maximum Nesting Depth

Implementations MUST enforce a maximum nesting depth of 8 levels. If a lookup exceeds 8 levels of nesting, the implementation MUST:

1. Abort the lookup
2. Return an error (SOPHIA_EVT_MISS)
3. Emit an EVENT_ANOMALY to Anamnesis
4. Use the default value for the field

This prevents infinite loops from circular references in the dictionary graph.

4.2.6. Circular Reference Detection

Implementations MUST detect circular references during lookup. A circular reference occurs when a nested lookup revisits a sub-dictionary index that was previously visited in the same lookup chain.

Detection is performed by maintaining a visited set (bitmask of up to 256 sub-dictionary indices) during each lookup:

```
visited = 0  // 256-bit bitmask
for each level:
  if visited & BIT(sub_dict_id):
    // Circular reference detected
    abort_lookup()
    emit EVENT_ANOMALY
    return default_value
  visited |= BIT(sub_dict_id)
  proceed_to_next_level()
```

4.2.7. Use Cases for Hierarchical Dictionaries

1. *Service Topology*: Model service dependencies as a tree. Root -> service_group -> service_instance -> endpoint.
2. *Policy Hierarchies*: Inherit QoS policies from parent categories. Root -> department -> team -> service -> policy.
3. *Geographic Routing*: Organize routing prefixes by region. Root -> continent -> country -> datacenter -> rack.

4. *Tenant Isolation*: Multi-tenant deployments where each tenant has its own sub-dictionary namespace. Root -> tenant_id -> service_identity -> endpoint.

4.2.8. CBOR Encoding for Nested Entries

Nested sub-dictionary entries are serialized in CBOR as:

```
typed_sub_entry = {
  "entry_type": uint,              ; 0=LEAF, 1=BRANCH, 2=COMPOSITE, 3=ALIAS
  "name": tstr,                    ; Human-readable name
  ? "endpoint": tstr,              ; Service endpoint (LEAF/COMPOSITE)
  ? "pqc_algorithm": uint,         ; PQC algorithm ID (LEAF/COMPOSITE)
  ? "pqc_pubkey": bstr,            ; Public key bytes (LEAF/COMPOSITE)
  ? "pqc_fingerprint": bstr,       ; SHA3-256 truncation (LEAF/COMPOSITE)
  ? "key_epoch": uint,             ; Rotation counter (LEAF/COMPOSITE)
  ? "key_expires": tstr,           ; ISO 8601 timestamp (LEAF/COMPOSITE)
  ? "nested_dict_id": uint,        ; Nested sub-dict index (BRANCH/COMPOSITE)
  ? "alias_target": [uint, uint],  ; [sub_dict_id, sub_key] (ALIAS only)
  ? "description": tstr,           ; Additional metadata
}
```

4.2.9. BPF Map Representation for Nested Dictionaries

Nested dictionaries use the same BPF_MAP_TYPE_ARRAY_OF_MAPS indirection as top-level sub-dictionaries. The sophia_dicts array is shared between top-level and nested sub-dictionaries:

```
sophia_dicts[0-63]:    Reserved for top-level sub-dictionaries
sophia_dicts[64-191]:  Available for nested sub-dictionaries
sophia_dicts[192-255]: Reserved for future use
```

Nested sub-dictionary indices MUST be in the range [64-191]. Top-level sub-dictionary indices MUST be in the range [0-63]. This partitioning prevents accidental conflicts between top-level and nested dictionaries.

4.3. Root Dictionary

The root dictionary is a single BPF hash map with 256 slots. Each key (0x00-0xFF) maps to a root entry structure.

Root entries occupy the following key ranges:

* 0x00: Reserved (MUST NOT be used by any implementation)

* 0x01-0x0F: Standard categories (see Section 8)

* 0x10-0xFE: Available for operator assignment

* 0xFF: Reserved (Yaldabaoth chaos injection)

4.4. Initialization Guarantee

The root dictionary MUST be fully initialized before any Monad packets are processed by Shim programs. Wotan [WOTAN] or system initialization logic MUST:

1. Load all root entries from persistent storage

2. Verify that each standard root key (0x01-0x06) has a corresponding entry

3. Initialize default values for any missing entries using base=2, multiplier=1

4. Signal readiness to shield/shim components only after this initialization is complete

Any attempt to process a packet before Sophia is initialized is a fatal configuration error and MUST be logged.

4.5. Dictionary Size Constraints

Dictionary capacity is bounded by both entry count and total byte size to prevent denial-of-service attacks via unbounded memory exhaustion.

4.5.1. Per-Flow Dictionary Capacity

Each flow maintains its own dictionary with strict limits:

* Maximum 128 entries per flow

* Maximum 1 MB total size per flow

When adding a new entry would exceed either limit: - Reject the new entry - Return error code 0x09 (Insufficient buffer space) - Do NOT evict existing entries - Emit audit event with reason "dictionary_full"

4.5.2. Global Dictionary Capacity

System-wide dictionary capacity MUST NOT exceed 100 MB.

5. QPACK Compression Headers for Dictionary Entries (NEW in draft-03)

5.1. Motivation

Sophia dictionary entries can be large, especially when carrying PQC key material (ML-KEM-768 public keys are 1184 bytes, ML-DSA-65 public keys are 1952 bytes). Distributing full dictionary entries over Wotan topics [WOTAN] at cluster-wide scale incurs significant bandwidth.

QPACK (RFC 9204) is a header compression format designed for HTTP/3 that provides efficient encoding of key-value pairs with static and dynamic table references. This section adapts QPACK for Sophia dictionary entry compression.

5.2. QPACK Adaptation for Sophia

5.2.1. Static Table

The Sophia QPACK static table contains pre-defined entries for frequently-used dictionary field names and values:

Index	Name	Value
0	type	service_identity
1	type	flow_action
2	type	qos_class
3	type	deploy_ring
4	type	circuit_state
5	type	mesh_flags
6	name	(empty)
7	endpoint	(empty)
8	pqc_algorithm	1 (ML-KEM-768)
9	pqc_algorithm	3 (ML-DSA-65)
10	key_epoch	(empty)
11	key_expires	(empty)
12	entry_type	0 (LEAF)
13	entry_type	1 (BRANCH)
14	entry_type	2 (COMPOSITE)
15	entry_type	3 (ALIAS)
16	description	(empty)
17	nested_dict_id	(empty)
18	base	2
19	multiplier	1
20	unit	microseconds
21	unit	milliseconds
22	unit	nanoseconds
23	unit	packets

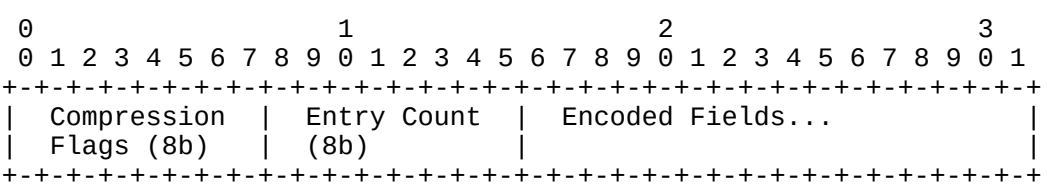
5.2.2. Dynamic Table

Each Wotan subscriber maintains a per-connection dynamic table for Sophia entries. The dynamic table is populated as dictionary updates are received and provides reference-based encoding for subsequent updates.

Dynamic table capacity: 4096 bytes (configurable via SETTINGS).

5.2.3. Encoding Format

Sophia QPACK-compressed entries use the following wire format:



Compression Flags: Bit 7: QPACK enabled (1=compressed, 0=raw CBOR)

Bit 6: Static table reference present

Bit 5: Dynamic table reference present

Bit 4: Huffman encoding used for string values

Bits 3-0: Reserved (MUST be zero)

Entry Count: Number of field entries in this compressed block (0-255).

Encoded Fields: QPACK-encoded field entries per RFC 9204 Section 4.5.

5.2.4. Encoding Rules

1. Fields with values matching static table entries MUST use static table references (1-byte encoding vs. full field encoding).

2. Fields with values matching dynamic table entries SHOULD use dynamic table references.

3. String values longer than 16 bytes SHOULD use Huffman encoding per RFC 9204.

4. PQC key material (pqc_pubkey, signature_pubkey) SHOULD NOT be Huffman-encoded (binary data has poor Huffman compression ratios).

5. New field values MUST be inserted into the dynamic table for future reference, unless the dynamic table is full.

5.2.5. Compression Ratio

Expected compression ratios for common Sophia entries:

Entry Type	Raw CBOR	QPACK	Ratio
service_identity (no PQC)	~120 bytes	~40 bytes	3:1
service_identity (with PQC)	~1400 bytes	~1250 bytes	1.1:1
flow_action	~60 bytes	~12 bytes	5:1
qos_class	~50 bytes	~10 bytes	5:1
typed_sub_entry (BRANCH)	~80 bytes	~25 bytes	3:1
typed_sub_entry (COMPOSITE)	~200 bytes	~60 bytes	3:1

PQC key material is incompressible; compression primarily benefits metadata fields (names, types, endpoints).

5.2.6. Decompression Limits

To prevent decompression bomb attacks:

1. Maximum decompressed entry size: 1 MB (per Section 3.3)

2. Maximum decompression time: 10 milliseconds

3. Maximum dynamic table size: 4096 bytes (prevents memory exhaustion)

4. Decompression MUST fail-safe: on error, reject entry with SOPHIA_EVT_DECOMP_FAIL and emit EVENT_ANOMALY

5.2.7. Backward Compatibility

QPACK compression is OPTIONAL. Dictionary entries with Compression Flags bit 7 = 0 are raw CBOR and are processed identically to draft-02 entries. Implementations that do not support QPACK MUST:

1. Check Compression Flags bit 7

2. If bit 7 = 1: reject entry with error SOPHIA_EVT_DECOMP_FAIL

3. If bit 7 = 0: process as raw CBOR (draft-02 compatible)

This ensures that draft-02 and draft-03 implementations can coexist during rolling upgrades.

6. Dictionary Distribution

Sophia dictionaries are distributed to all nodes via the Wotan [WOTAN] publish-subscribe topics. The distribution model ensures atomic, cluster-wide updates with zero packet loss.

6.1. Wotan Distribution Channel

Dictionaries are published on a versioned topic:

Topic: sophia.dictionary.v{N}

Where N = version number (0-255).

Each topic publication contains:
1. Complete serialized dictionary (CBOR or QPACK-compressed CBOR)
2. Version number (repeated for idempotence)
3. Timestamp (ISO 8601)
4. Signature (ML-DSA-65, optional for integrity)
5. Compression Flags (indicating whether QPACK is used) (NEW in draft-03)

6.2. Version Negotiation

Implementations MUST support at least 2 concurrent dictionary versions. When a new version is published:

1. Subscriber receives dictionary on sophia.dictionary.v{N+1}

2. New maps are created (sophia_dict_1_v{N+1}, etc.)

3. Old version maps remain active

4. Array-of-maps references are atomically updated

5. Old version maps are retained for grace_period (default: 60 seconds)

6. After grace_period, old maps are deleted

6.3. Atomic Update Protocol

The update sequence is:

1. [Provisioning] Publish new dictionary to sophia.dictionary.v{N+1}
2. [Wotan] Receive on subscriber
3. [Wotan] Deserialize CBOR (decompress if QPACK) -> in-memory dictionary
4. [Wotan] Create new BPF maps with suffix _v{N+1}
5. [Wotan] Load all entries into new maps (including nested sub-dicts)
6. [Wotan] Update sophia_dicts[0..255] pointers atomically
   (single atomic map write per sub-dict)
7. [Wotan] Update sophia_version map (key 0, value = N+1)
8. [Wotan] Retain old maps for grace_period
9. [Wotan] After grace_period, delete old maps

   All Shim/Shield nodes see the update within one polling cycle
   (typically <10ms) of the Wotan write.

7. Security Considerations

7.1. Dictionary Poisoning Attack Vectors

   Dictionary poisoning attacks attempt to corrupt semantic metadata,
   enabling: - Service misidentification (0x01 maps to wrong endpoint) -
   Policy bypass (0x02 maps to permissive action instead of drop) -
   Cache invalidation (Shim programs crash on bad dictionary entry)

   Defense mechanisms are specified throughout this document:

   1. *ML-DSA-65 Signature Verification*: All entries from provisioning
      nodes MUST be signed with ML-DSA-65.

   2. *Timestamp Validation*: Reject updates with timestamps > 5
      minutes in future/past.

   3. *CRC32 Integrity Checks*: CBOR payloads include CRC32 checksums.

   4. *Source Authentication*: Whitelist of provisioning node public
      keys.

7.2. Nested Dictionary Security

7.2.1. Depth Limit Enforcement

   The maximum nesting depth of 8 levels MUST be enforced in all lookup
   paths. Exceeding this limit is treated as a security violation
   (potential DoS via deeply nested dictionaries).

7.2.2. Circular Reference Prevention

Circular reference detection (Section 3.2.5) MUST be performed on every nested lookup. Circular references in dictionary data could cause infinite loops in BPF programs, leading to verifier timeouts or denial of service.

7.2.3. Namespace Partitioning

The partitioning of sophia_dicts indices (0-63 for top-level, 64-191 for nested, 192-255 reserved) MUST be enforced. An attacker who can control nested_dict_id values could reference top-level dictionaries as nested dictionaries, causing semantic confusion.

7.3. QPACK Decompression Security

7.3.1. Decompression Bomb Mitigation

QPACK-compressed dictionary entries are subject to the same decompression limits as draft-02 compressed entries:

1. Maximum decompressed output: 1 MB
2. Maximum decompression time: 10 milliseconds
3. Explicit compression flags (no implicit detection)

7.3.2. Dynamic Table Poisoning

An attacker could send dictionary updates designed to fill the dynamic table with malicious entries, causing subsequent legitimate entries to reference attacker-controlled values.

Mitigation: - Dynamic table entries MUST be validated against Sophia schema - Dynamic table MUST be flushed on dictionary version change - Dynamic table capacity is limited to 4096 bytes

7.4. Cross-Reference with Foundation and Wotan

Security considerations in this memo are aligned with:

1. *[UNHEADED-FOUNDATION] Section 10 - Security Considerations*: Wire format immutability threat model, parser divergence attacks, and verification procedures.
2. *[WOTAN] Section 12 - Security Considerations*: Topic injection attacks, ring buffer memory exhaustion, cross-flow memory access, and WAL tampering detection.

8. PQC Key Dictionary Integration (NEW in draft-03 update)

8.1. Overview

Sophia provides the key management layer for the post-quantum cryptographic (PQC) authentication system defined in [UNHEADED-FOUNDATION]. Full PQC signatures and public keys are stored in Sophia BPF maps, while the Monad wire format carries only 12-byte references (see [UNHEADED-FOUNDATION] PQC Authentication Value Format).

8.2. PQC Signature Map (PQC_SIG_MAP)

The PQC_SIG_MAP is a BPF hash map keyed by the 32-bit SigRef value from the PQC authentication value.

```
struct {
    __uint(type, BPF_MAP_TYPE_HASH);
    __uint(max_entries, 65536);
    __type(key, u32);        // SigRef from PQC value
    __type(value, struct pqc_sig_entry);
} pqc_sig_map SEC(".maps");

struct pqc_sig_entry {
    u8   algo_id;            // PQC algorithm (0x01-0x05)
    u8   status;             // Verification status
    u16  key_ref;            // Cross-reference to PQC_KEY_MAP
    u32  sig_len;            // Signature length in bytes
    u8   hash_pfx[4];        // SHA-256(signature)[0:4]
    u8   signature[];        // Variable-length signature data
};
```

8.3. PQC Key Map (PQC_KEY_MAP)

The PQC_KEY_MAP stores public keys indexed by the 16-bit KeyRef value.

```
struct {
    __uint(type, BPF_MAP_TYPE_HASH);
    __uint(max_entries, 4096);
    __type(key, u16);        // KeyRef from PQC value
    __type(value, struct pqc_key_entry);
} pqc_key_map SEC(".maps");

struct pqc_key_entry {
    u8   algo_id;            // PQC algorithm identifier
    u8   key_epoch;          // Key rotation counter
    u16  reserved;
    u32  key_len;            // Public key length in bytes
    u8   fingerprint[32];    // SHA3-256 of public key
    u8   expires[8];         // Expiry timestamp (Unix ns, BE)
    u8   pubkey[];           // Variable-length public key data
};
```

8.4. PQC Dictionary Operations

8.4.1. Signature Lookup

When a BPF program verifies a PQC authentication value:

1. Extract SigRef (u32) from Monad scratch bytes [0x0E..0x11]
2. Look up SigRef in pqc_sig_map
3. Compare hash_pfx with SHA-256(sig_entry.signature)[0:4]
4. If match: use cached verification status
5. If miss or mismatch: emit EVENT_ANOMALY, use default policy

8.4.2. Key Rotation

Key rotation is managed through the key_epoch counter:

1. New key published to sophia.pqc.keys.v{N+1} Wotan topic
2. Subscribers create new pqc_key_map entries with epoch+1
3. Old keys retained for grace_period (default: 300 seconds)
4. After grace_period: old key entries deleted
5. Signatures referencing expired keys receive status=Expired (0x03)

8.5. PQC Algorithm Support Matrix

Algo ID	Algorithm	Key Size	Sig Size	Use Case
0x01	SLH-DSA	32-64 B	7856-49856 B	Hash-based (conservative)
0x02	ML-DSA	1312-2592 B	2420-4627 B	Lattice-based (standard)
0x03	FN-DSA	897-1793 B	666-1280 B	Lattice-based (compact)
0x04	ML-KEM	800-1568 B	768-1568 B	Key encapsulation
0x05	HQC	2249-7245 B	4497-14469 B	Code-based (conservative)

Implementations MUST support at least ML-DSA (0x02) and SHOULD support SLH-DSA (0x01) for defense-in-depth.

9. UPC Opcode Dictionary for Sophia-Driven Instruction Decode (NEW in draft-03 update)

9.1. Overview

The UPC compute engine uses Sophia dictionaries for instruction decode, enabling runtime-reconfigurable instruction semantics. Instead of hardcoding opcode meanings, the MBC ISA opcodes are mapped through a Sophia dictionary that provides human-readable names, execution metadata, and instruction class information.

9.2. Opcode Dictionary Structure

The opcode dictionary is a Sophia sub-dictionary (root key 0x10, "code" category) that maps 8-bit opcodes to instruction metadata.

```
Root entry 0x10 -> "code" -> sub-dict #16
  Sub-dict #16[0x00] -> {name: "NOP", class: "control", cycles: 1}
  Sub-dict #16[0x01] -> {name: "ADD", class: "arithmetic", cycles: 1}
  Sub-dict #16[0x30] -> {name: "LD", class: "memory", cycles: 2}
  Sub-dict #16[0x40] -> {name: "SYSCALL", class: "system", cycles: 0}
```

9.3. Instruction Class Types

Class	Value	Description
arithmetic	0x01	ALU operations (ADD, SUB, MUL, DIV, etc.)
logical	0x02	Bitwise operations (AND, OR, XOR, NOT, shifts)
stack	0x03	Stack operations (PUSH, POP)
register	0x04	Register operations (MOV, MOVI, CMP)
branch	0x05	Control flow (JMP, JZ, CALL, RET, etc.)
memory	0x06	Memory access (LD, ST, LDB, STB, etc.)
atomic	0x07	Atomic operations (CLI, STI, XCHG, CAS)
system	0x08	System operations (SYSCALL, HALT)
interrupt	0x09	Interrupt handling (INT, IRET)
extended	0x0A	Extended operations (LOAD_IMM32, ADDI, etc.)

9.4. BPF Map Representation

```
struct sophia_opcode_entry {
    u8   opcode;              // MBC opcode value
    u8   insn_class;          // Instruction class (0x01-0x0A)
    u8   base_cycles;         // Base cycle count (1-4)
    u8   flags;               // Bit 0: modifies flags
                              // Bit 1: reads memory
                              // Bit 2: writes memory
                              // Bit 3: modifies PC
    u8   name[24];            // Null-terminated mnemonic
    u32  reserved;            // Reserved (MUST be zero)
};  // Total: 32 bytes per entry (matches SophiaEntry size)
```

9.5. Sophia-Driven Decode in BPF

During instruction execution, the MBC CPU MAY look up the opcode in the Sophia opcode dictionary for:

1. *Instruction tracing*: Emit human-readable instruction name in Anamnesis events instead of raw opcode values

2. *Dynamic dispatch*: Runtime-reconfigurable instruction behavior via dictionary updates (experimental)

3. *Profiling*: Per-instruction-class cycle counting and metrics

4. *Validation*: Verify opcode is in the valid set before execution

Implementations MAY cache opcode dictionary entries in a per-CPU BPF array map for fast lookup (~50 ns vs. ~150 ns for hash lookup).

10. IANA Considerations

10.1. Sophia Root Key Registry

IANA SHOULD establish a new registry:

```
   Registry Name:  Unheaded Sophia Root Dictionary Keys
   Template:       Root Key (0x00-0xFF), Category Name, Type,
                   Specification Reference
   Policy:         0x00-0x0F: Specification Required
                   0x10-0xFE: First Come First Served
                   0xFF: Specification Required (reserved)

   Initial entries:
     0x00: RESERVED
     0x01: service_identity
     0x02: flow_action
     0x03: qos_class
     0x04: deploy_ring
     0x05: circuit_state
     0x06: mesh_flags
     0x07-0x0F: RESERVED for future standardization
     0x10: routing (routing entry type)
     0x11: firewall (firewall entry type)
     0x12: observability (observability entry type)
     0x13: ids (IDS entry type)
     0x14: health (health entry type)
     0xFF: RESERVED (Yaldabaoth)
```

10.2. Sophia Sub-Dictionary Type Registry (NEW in draft-03)

```
   IANA SHOULD establish a new registry:

   Registry Name:  Unheaded Sophia Sub-Dictionary Types
   Template:       Type Code (0x00-0xFF), Type Name, Description,
                   Specification Reference
   Policy:         Specification Required

   Initial entries:
     0x00: LEAF (leaf node, metadata only)
     0x01: BRANCH (branch node, nested sub-dictionary reference)
     0x02: COMPOSITE (metadata + nested sub-dictionary)
     0x03: ALIAS (indirection to another entry)
     0x04-0xFF: Reserved for future use
```

10.3. Sophia QPACK Static Table Registry (NEW in draft-03)

```
   Registry Name:  Unheaded Sophia QPACK Static Table Entries
   Template:       Index (uint), Field Name, Default Value,
                   Specification Reference
   Policy:         Specification Required

   Initial entries: See Section 5.2.1 (24 entries, indices 0-23)
```

11. Author’s Address

Stevie Bellis Unheaded Email: stevie@bellis.tech

12. References

12.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/rfc/rfc2119>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

[RFC8610] Birkholz, H., Vigano, C., and C. Bormann, "Concise Data Definition Language (CDDL): A Notational Convention to Express Concise Binary Object Representation (CBOR) and JSON Data Structures", RFC 8610, DOI 10.17487/RFC8610, June 2019, <https://www.rfc-editor.org/rfc/rfc8610>.

[RFC8949] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, RFC 8949, DOI 10.17487/RFC8949, December 2020, <https://www.rfc-editor.org/rfc/rfc8949>.

[RFC9204] Krasic, C., Bishop, M., and A. Frindell, Ed., "QPACK: Field Compression for HTTP/3", RFC 9204, DOI 10.17487/RFC9204, June 2022, <https://www.rfc-editor.org/rfc/rfc9204>.

[RFC9669] Thaler, D., Ed., "BPF Instruction Set Architecture (ISA)", RFC 9669, DOI 10.17487/RFC9669, October 2024, <https://www.rfc-editor.org/rfc/rfc9669>.

[FIPS204] NIST, "Module-Lattice-Based Digital Signature Standard", August 2024.

[UNHEADED-FOUNDATION] Bellis, S., "Unheaded: Protocol Foundation", Work in Progress, Internet-Draft, draft-bellis-unheaded-protocol-foundation-00, March 2026, <https://datatracker.ietf.org/doc/html/draft-bellis-unheaded-protocol-foundation-00>.

[WOTAN] Bellis, S., "Wotan Memory Protocol for the Unheaded Protocol", Work in Progress, Internet-Draft, draft-bellis-unheaded-wotan-memory-00, March 2026, <https://datatracker.ietf.org/doc/html/draft-bellis-unheaded-wotan-memory-00>.

12.2. Informative References

[RFC0768] Postel, J., "User Datagram Protocol", STD 6, RFC 768, DOI 10.17487/RFC0768, August 1980, <https://www.rfc-editor.org/rfc/rfc768>.

[RFC0791] Postel, J., "Internet Protocol", STD 5, RFC 791, DOI 10.17487/RFC0791, September 1981, <https://www.rfc-editor.org/rfc/rfc791>.

[RFC0792] Postel, J., "Internet Control Message Protocol", STD 5, RFC 792, DOI 10.17487/RFC0792, September 1981, <https://www.rfc-editor.org/rfc/rfc792>.

[RFC0793] Postel, J., "Transmission Control Protocol", RFC 793, DOI 10.17487/RFC0793, September 1981, <https://www.rfc-editor.org/rfc/rfc793>.

[RFC8799] Carpenter, B. and B. Liu, "Limited Domains and Internet Protocols", RFC 8799, DOI 10.17487/RFC8799, July 2020, <https://www.rfc-editor.org/rfc/rfc8799>.

[RFC9197] Brockners, F., Ed., Bhandari, S., Ed., and T. Mizrahi, Ed., "Data Fields for In Situ Operations, Administration, and Maintenance (IOAM)", RFC 9197, DOI 10.17487/RFC9197, May 2022, <https://www.rfc-editor.org/rfc/rfc9197>.

Appendix A. Changes from draft-bellis-unheaded-sophia-dictionary-02

The following changes are made in draft-03:

1. *Sub-Dictionary Type System (NEW)*: Added Section 3.2 defining typed sub-dictionary entries (LEAF, BRANCH, COMPOSITE, ALIAS) for hierarchical knowledge representation. Includes nested lookup chains, maximum nesting depth (8 levels), circular reference detection, BPF map representation, CBOR encoding, and use cases.

2. *QPACK Compression Headers (NEW)*: Added Section 5 defining QPACK-based compression for dictionary entries. Includes static table (24 entries), dynamic table management, encoding format, compression ratios, decompression limits, and backward compatibility with draft-02 raw CBOR entries.

3. *Cross-References to Foundation draft-06 (UPDATED)*: Updated UNHEADED-FOUNDATION reference from draft-04 to draft-06. Added Section 1.1 documenting the specification family structure. Added cross-references to Wotan draft-03 throughout.

4. *Sophia Sub-Dictionary Type Registry (NEW IANA)*: Added IANA registry for sub-dictionary type codes (0x00-0xFF).

5. *Sophia QPACK Static Table Registry (NEW IANA)*: Added IANA registry for QPACK static table entries.

6. *Nested Dictionary Security (NEW)*: Added security considerations for depth limit enforcement, circular reference prevention, namespace partitioning, QPACK decompression bomb mitigation, and dynamic table poisoning.

7. *PQC Key Dictionary Integration (NEW)*: Added PQC_SIG_MAP and PQC_KEY_MAP BPF map definitions for storing full post-quantum signatures and public keys. Defines signature lookup, key rotation protocol, and algorithm support matrix covering SLH-DSA, ML-DSA, FN-DSA, ML-KEM, and HQC.

8. *UPC Opcode Dictionary (NEW)*: Added Sophia-driven instruction decode for the MBC ISA. Defines opcode dictionary structure (root key 0x10, "code" category), 10 instruction class types, 32-byte BPF map entry format, and Sophia-driven decode use cases (tracing, dynamic dispatch, profiling, validation).

9. *Updated Date*: Changed date from 2026-03-05 to 2026-03-15.

All changes in draft-03 are purely additive. No existing dictionary format, wire encoding, or processing rule from draft-02 is modified or removed. Draft-02 CBOR entries remain valid in draft-03.

Appendix B. Acknowledgments

The Linux kernel BPF community (Alexei Starovoitov, Daniel Borkmann, Song Liu) for the infrastructure enabling per-packet computation in the kernel datapath.

The authors of RFC 9669 (BPF ISA), RFC 8799 (Limited Domains), and RFC 9204 (QPACK) for the foundational protocols that make this design possible.

This document was co-authored with assistance from Claude (Anthropic).

Author’s Address

Stevie Bellis
Unheaded
United States of America
Email: stevie@bellis.tech