OERC-S v0.1: Orbital Energy Routing & Clearing Standard

OERC-S defines the payable energy receipt for machine-to-machine settlement: the COLLAPSE.

Version: 0.1 **Status:** Draft **Date:** 2025-12-27 **Author:** Adelio Sebti **License:** CC-BY-4.0

In 30 Seconds

What: A cryptographic receipt proving energy was delivered and settled.

Why: When satellites beam power to ground stations, or when AGI/ASI systems trade energy autonomously, every actor needs a universal, machine-verifiable proof of payment.

How:

1. **INTENT** = "I want X joules" (non-payable)
2. **FRAME** = "Energy transfer in progress" (non-payable)
3. **COLLAPSE** = "Done. Here's the proof." → **Only payable object**

Optimized for:

- Machine-to-machine settlement (AGI, ASI, autonomous systems)
- Space-based solar power (SBSP)
- Power-beaming networks
- Orbital compute and relays
- Any multi-actor energy system

Your action: Publish ONE Collapse hash from a real test.

Table of Contents

---

1. Abstract

OERC-S (Orbital Energy Routing & Clearing Standard) defines a minimal, object-first protocol for routing, proving, and settling energy transfers in space-based power infrastructure. The protocol operates on a fundamental axiom: **COLLAPSE is payable, everything else is not.** INTENTs represent pre-allocation reservations that carry no settlement obligation; FRAMEs represent superposed streaming proofs that remain probabilistic until finalization; only COLLAPSE objects—emitted at clearing window boundaries—constitute binding, payable settlement records. This design ensures that streaming telemetry and routing negotiations impose zero settlement risk while maintaining cryptographic auditability, enables efficient netting across multi-hop relay topologies, and provides a clear demarcation between operational signaling and financial obligation. OERC-S is transport-agnostic, crypto-agile, and designed for deterministic verification across heterogeneous node implementations.

Quick Summary

Problem	Multi-actor energy systems have no common payable artifact for machine-to-machine settlement
Solution	Three objects: INTENT → FRAME → COLLAPSE. **Only COLLAPSE is payable**
Action	Publish ONE real Collapse hash from your test environment
Not in scope	Routing, transmission, hardware — OERC-S settles receipts only

```
INTENT (non-payable) → FRAME* (non-payable) → COLLAPSE (payable)
```

1.1 Partner Action (Minimal)

To demonstrate conformance, publish these fields from ONE real Collapse:

Field	Description
`collapse_id`	BLAKE3 hash identifying the collapse
`window_id`	Clearing window identifier
`net_energy_j`	Net energy transferred (joules)
`merkle_root`	Merkle root of frames in window

This enables third-party verification without revealing proprietary telemetry.

---

2. Terminology

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 RFC 2119.

2.1 Core Terms

Node: Any entity capable of transmitting, receiving, relaying, or storing energy within the OERC-S network. Nodes are identified by public keys.

Energy Transfer: The physical movement of energy between nodes via any supported modality (laser, microwave, storage handoff, relay).

Intent: A cryptographically signed declaration of willingness to participate in an energy transfer within specified constraints. Non-binding until collapsed.

Frame: A streaming proof object representing partial or in-progress energy transfer. Frames are superposed—they represent probabilistic state until collapse.

Collapse: The finalization event that converts superposed frames into a single, deterministic, payable settlement record.

Window: A discrete time interval during which frames accumulate and at whose boundary collapse occurs.

Timebox: A temporal constraint defined by start and end timestamps in ISO 8601 format.

Modality: The physical mechanism of energy transfer: `laser`, `microwave`, `storage`, or `relay`.

Segment: A single hop within a multi-hop energy transfer, representing transmission from one node to another.

Netting: The process of aggregating and offsetting energy flows to produce net settlement obligations.

Canonical Form: The deterministic, reproducible encoding of an object for hashing and signature operations.

2.2 Cryptographic Terms

Crypto Suite: A named collection of algorithms for signing, verification, key exchange, and hashing.

Sigset: An array of signatures over an object's canonical form, potentially from multiple signers using multiple crypto suites.

Commitment Hash: A cryptographic commitment to data that can be revealed later for verification.

Finality Proof: A cryptographic attestation that a collapse is valid and irreversible within protocol rules.

2.3 Identifiers

intent_id: BLAKE3 hash of the INTENT's canonical encoding. 32 bytes.

frame_id: BLAKE3 hash of the FRAME's canonical encoding. 32 bytes.

collapse_id: BLAKE3 hash of the COLLAPSE's canonical encoding. 32 bytes.

window_id: Deterministic identifier for a clearing window, derived from window parameters.

pubkey: Public key identifying a node, encoded per the relevant crypto suite.

2.4 Units

Joule (J): The SI unit of energy. All energy quantities in OERC-S are denominated in joules.

Basis Point (bp): One hundredth of one percent (0.01%). Used for fee specification.

---

3. Object Definitions

n``` INTENT (non-payable) → FRAME* (superposed, non-payable) → COLLAPSE (payable)

```
OERC-S defines three primary object types, each serving a distinct role in the energy routing and clearing lifecycle. Objects
are immutable once created; modifications require new objects with new identifiers.

### 3.1 INTENT

An INTENT represents a pre-allocation or reservation of energy transfer capacity between nodes. INTENTs are negotiation
artifacts—they express willingness and constraints but carry no settlement obligation.

#### 3.1.1 INTENT Fields

| Field | Type | Description |
|-------|------|-------------|
| `intent_id` | bytes[32] | BLAKE3 hash of canonical encoding (computed, not transmitted) |
| `version` | string | Protocol version. MUST be "OERC-S/0.1" |
| `tx_node_pubkey` | bytes | Public key of the transmitting (source) node |
| `rx_node_pubkey` | bytes | Public key of the receiving (destination) node |
| `timebox` | object | Temporal constraints for the intent |
| `timebox.t_start` | string | ISO 8601 timestamp for validity start |
| `timebox.t_end` | string | ISO 8601 timestamp for validity end |
| `max_energy_j` | uint64 | Maximum energy in joules for this intent |
| `modality_set` | array[string] | Acceptable transfer modalities |
| `location_tag` | object | Spatial constraints and identifiers |
| `location_tag.orbit_regime` | string | Orbital regime identifier (e.g., "LEO", "GEO", "L1") |
| `location_tag.ephemeris_hash` | bytes[32] | BLAKE3 hash of ephemeris data |
| `location_tag.footprint_id` | string | Ground footprint identifier, if applicable |
| `crypto_suite_id` | string | Identifier of the crypto suite used for signatures |
| `sigset` | array[object] | Array of signatures over canonical form |
| `created_at` | string | ISO 8601 timestamp of intent creation |
| `metadata` | object | Optional application-specific metadata |

#### 3.1.2 INTENT Semantics

1. An INTENT is valid if and only if:
   - All required fields are present and well-formed
   - `t_start` < `t_end`
   - `max_energy_j` > 0
   - `modality_set` contains at least one valid modality
   - At least one signature in `sigset` is valid over the canonical form
   - The signing key matches either `tx_node_pubkey` or `rx_node_pubkey`

2. An INTENT becomes **active** when the current time >= `t_start` and < `t_end`.

3. An INTENT expires when the current time >= `t_end`. Expired INTENTs MUST NOT be used for new FRAMEs.

4. Multiple INTENTs MAY exist between the same node pair with overlapping timeboxes. Resolution is application-specific.

5. An INTENT does not guarantee energy will be transferred. It only establishes parameters for potential transfer.

#### 3.1.3 INTENT Example (JSON)

```json
{
  "version": "OERC-S/0.1",
  "tx_node_pubkey": "b4a3f2e1d0c9b8a7f6e5d4c3b2a190817060504030201 00f0e0d0c0b0a090807",
  "rx_node_pubkey": "010203040506070809 0a0b0c0d0e0f10111213141516171819 1a1b1c1d1e1f20",
  "timebox": {
    "t_start": "2025-12-27T00:00:00Z",
    "t_end": "2025-12-27T01:00:00Z"
  },
  "max_energy_j": 1000000000,
  "modality_set": ["laser", "microwave"],
  "location_tag": {
    "orbit_regime": "LEO",
```
```

```
    "ephemeris_hash": "a1b2c3d4e5f6a1b2c3d4e5f6a1b2c3d4e5f6a1b2c3d4e5f6a1b2c3d4e5f6a1b2",
    "footprint_id": "NA-WEST-001"
  },
  "crypto_suite_id": "ED25519-BLAKE3",
  "sigset": [
    {
      "signer_pubkey": "b4a3f2e1d0c9b8a7f6e5d4c3b2a190817060504030201 00f0e0d0c0b0a090807",
      "signature": "...",
      "crypto_suite_id": "ED25519-BLAKE3"
    }
  ],
  "created_at": "2025-12-26T23:55:00Z",
  "metadata": null
}
```

3.2 FRAME

A FRAME represents superposed streaming proofs of energy transfer. FRAMEs are emitted during active energy transmission to provide real-time auditability without creating settlement obligations. FRAMEs reference an INTENT and contain one or more segments representing individual hops.

3.2.1 FRAME Fields

Field	Type	Description
`frame_id`	bytes[32]	BLAKE3 hash of canonical encoding (computed, not transmitted)
`version`	string	Protocol version. MUST be "OERC-S/0.1"
`intent_id`	bytes[32]	Reference to parent INTENT
`window_id`	bytes[32]	REQUIRED. Reference to clearing window (binds frame to window)
`seq_no`	uint64	Sequence number within the intent, starting at 0
`segments`	array[object]	Array of segment objects
`crypto_suite_id`	string	Identifier of the crypto suite used for signatures
`sigset`	array[object]	Array of signatures over canonical form
`created_at`	string	ISO 8601 timestamp of frame creation
`metadata`	object	Optional application-specific metadata

3.2.2 Segment Fields

Each segment within a FRAME represents a single hop in the energy transfer path.

Field	Type	Description
`segment_id`	uint32	Segment index within this frame, starting at 0
`from_pubkey`	bytes	Public key of the transmitting node for this segment
`to_pubkey`	bytes	Public key of the receiving node for this segment
`modality`	string	Transfer modality for this segment
`energy_j_partial`	uint64	Energy transferred in this segment (joules)
`commitment_hash`	bytes[32]	Cryptographic commitment to segment data
`loss_estimate`	object	Estimated energy loss during transfer
`loss_estimate.min`	uint64	Minimum estimated loss (joules)
`loss_estimate.max`	uint64	Maximum estimated loss (joules)
`segment_timebox`	object	Temporal bounds for this segment
`segment_timebox.t_start`	string	ISO 8601 timestamp for segment start
`segment_timebox.t_end`	string	ISO 8601 timestamp for segment end

3.2.3 FRAME Semantics

1. A FRAME is valid if and only if:
    - All required fields are present and well-formed
    - `intent_id` references a valid, active INTENT
    - `seq_no` is greater than any previously observed `seq_no` for this intent
    - At least one segment is present
    - All segments are internally consistent (see Section 6.4)
    - At least one signature in `sigset` is valid over the canonical form
    - Segment timeboxes fall within the parent INTENT's timebox
2. FRAMEs are **superposed**—they represent probabilistic state. Multiple FRAMEs with the same `intent_id` and overlapping time ranges MAY coexist.
3. FRAMEs do not create settlement obligations. They are evidence, not invoices.
4. The sum of `energy_j_partial` across all segments in a FRAME SHOULD NOT exceed `max_energy_j` from the parent INTENT, but this is not a hard constraint (overdelivery is possible).
5. Segment ordering within a FRAME represents the physical path of energy flow. `segments[0].from_pubkey` SHOULD match the INTENT's `tx_node_pubkey`.

3.2.4 FRAME Example (JSON)


```
{
  "version": "OERC-S/0.1",
  "intent_id": "c3d4e5f6a7b8c9d0e1f2a3b4c5d6e7f8a9b0c1d2e3f4a5b6c7d8e9f0a1b2c3d4",
  "seq_no": 42,
  "segments": [
    {
      "segment_id": 0,
      "from_pubkey": "b4a3f2e1d0c9b8a7f6e5d4c3b2a190817060504030201 00f0e0d0c0b0a090807",
      "to_pubkey": "5566778899aabbccddeeff00112233445566778899aabbccddeeff0011223344",
      "modality": "laser",
      "energy_j_partial": 500000000,
      "commitment_hash": "1234567890abcdef1234567890abcdef1234567890abcdef1234567890abcdef",
      "loss_estimate": {
        "min": 1000000,
        "max": 5000000
      },
      "segment_timebox": {
        "t_start": "2025-12-27T00:15:00Z",
        "t_end": "2025-12-27T00:15:30Z"
      }
    },
    {
      "segment_id": 1,
      "from_pubkey": "5566778899aabbccddeeff00112233445566778899aabbccddeeff0011223344",
      "to_pubkey": "0102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f20",
      "modality": "microwave",
      "energy_j_partial": 495000000,
      "commitment_hash": "fedcba0987654321fedcba0987654321fedcba0987654321fedcba0987654321",
      "loss_estimate": {
        "min": 2000000,
        "max": 8000000
      },
      "segment_timebox": {
        "t_start": "2025-12-27T00:15:30Z",
        "t_end": "2025-12-27T00:16:00Z"
      }
    }
  ],
  "crypto_suite_id": "ED25519-BLAKE3",
  "sigset": [
    {
      "signer_pubkey": "5566778899aabbccddeeff00112233445566778899aabbccddeeff0011223344",
      "signature": "...",
      "crypto_suite_id": "ED25519-BLAKE3"
    }
  ],
  "created_at": "2025-12-27T00:16:01Z",
```

```
  "metadata": null
}
```

3.3 COLLAPSE

A COLLAPSE is the finality object—the only object type that creates a binding, payable settlement obligation. COLLAPSE objects are emitted at clearing window boundaries and represent the deterministic resolution of all superposed FRAMEs within that window.

3.3.1 COLLAPSE Fields

Field	Type	Description
`collapse_id`	bytes[32]	BLAKE3 hash of canonical encoding (computed, not transmitted)
`version`	string	Protocol version. MUST be "OERC-S/0.1"
`intent_id`	bytes[32]	Reference to parent INTENT
`window_id`	bytes[32]	Identifier of the clearing window
`net_energy_j`	int64	Net energy transferred (signed; negative = reverse flow)
`net_energy_confidence`	object	Confidence bounds on net energy
`net_energy_confidence.min`	int64	Minimum net energy (joules)
`net_energy_confidence.max`	int64	Maximum net energy (joules)
`fees`	object	Fee specification for this collapse
`fees.type`	string	Fee type: "basis_points" or "absolute_j"
`fees.value`	uint64	Fee amount (basis points or joules)
`finality_proof`	object	Cryptographic proof of finality
`finality_proof.merkle_root`	bytes[32]	Merkle root of all frames in window
`finality_proof.signer_set`	array[bytes]	Public keys of finality attesters
`finality_proof.window_params`	object	Window parameters used for collapse
`finality_proof.signatures`	array[object]	Signatures from finality attesters
`crypto_suite_id`	string	Identifier of the crypto suite used
`sigset`	array[object]	Array of signatures over canonical form
`created_at`	string	ISO 8601 timestamp of collapse creation
`metadata`	object	Optional application-specific metadata

3.3.2 Finality Proof Structure

The `finality_proof` object provides cryptographic evidence that the collapse is valid and was produced correctly.

Field	Type	Description
`merkle_root`	bytes[32]	Root of Merkle tree over all frame_ids in window
`signer_set`	array[bytes]	Ordered list of attester public keys
`window_params`	object	Parameters defining the window
`window_params.window_start`	string	ISO 8601 timestamp for window start
`window_params.window_end`	string	ISO 8601 timestamp for window end
`window_params.window_duration_ms`	uint64	Window duration in milliseconds
`window_params.netting_algorithm`	string	Algorithm used for netting
`signatures`	array[object]	Signatures from signer_set members

3.3.3 COLLAPSE Semantics

1. A COLLAPSE is valid if and only if:

- All required fields are present and well-formed
- `intent_id` references a valid INTENT
- `window_id` is correctly derived from `window_params`
- `net_energy_confidence.min` <= `net_energy_j` <= `net_energy_confidence.max`
- `finality_proof` is valid (see Section 6.5)
- Sufficient signatures in `finality_proof.signatures` (threshold defined by application)
- At least one signature in `sigset` is valid over the canonical form

2. A COLLAPSE is **payable**. It represents a binding settlement obligation between parties.

3. Only one COLLAPSE MAY exist for a given (`intent_id` , `window_id`) pair. Duplicate collapses MUST be rejected.

4. COLLAPSE objects are immutable. Disputes are handled by creating new INTENTs and COLLAPSE objects, not by modifying existing ones.

5. The `net_energy_j` field is signed. Positive values indicate energy flow from `tx_node_pubkey` to `rx_node_pubkey` as specified in the INTENT. Negative values indicate reverse flow.

3.3.4 COLLAPSE Example (JSON)


```
{
  "version": "OERC-S/0.1",
  "intent_id": "c3d4e5f6a7b8c9d0e1f2a3b4c5d6e7f8a9b0c1d2e3f4a5b6c7d8e9f0a1b2c3d4",
  "window_id": "abcdef0123456789abcdef0123456789abcdef0123456789abcdef0123456789",
  "net_energy_j": 485000000,
  "net_energy_confidence": {
    "min": 480000000,
    "max": 490000000
  },
  "fees": {
    "type": "basis_points",
    "value": 25
  },
  "finality_proof": {
    "merkle_root": "9876543210fedcba9876543210fedcba9876543210fedcba9876543210fedcba",
    "signer_set": [
      "aabbccdd11223344aabbccdd11223344aabbccdd11223344aabbccdd11223344",
      "55667788aabbccdd55667788aabbccdd55667788aabbccdd55667788aabbccdd"
    ],
    "window_params": {
      "window_start": "2025-12-27T00:00:00Z",
      "window_end": "2025-12-27T01:00:00Z",
      "window_duration_ms": 3600000,
      "netting_algorithm": "SIMPLE_SUM_V1"
    },
    "signatures": [
      {
        "signer_pubkey": "aabbccdd11223344aabbccdd11223344aabbccdd11223344aabbccdd11223344",
        "signature": "...",
        "crypto_suite_id": "ED25519-BLAKE3"
      },
      {
        "signer_pubkey": "55667788aabbccdd55667788aabbccdd55667788aabbccdd55667788aabbccdd",
        "signature": "...",
        "crypto_suite_id": "ED25519-BLAKE3"
      }
    ]
  },
  "crypto_suite_id": "ED25519-BLAKE3",
  "sigset": [
    {
      "signer_pubkey": "b4a3f2e1d0c9b8a7f6e5d4c3b2a190817060504030201 00f0e0d0c0b0a090807",
      "signature": "...",
      "crypto_suite_id": "ED25519-BLAKE3"
    }
  ],
  "created_at": "2025-12-27T01:00:05Z",
  "metadata": null
}
```


---

4. Clearing Windows

Clearing windows provide temporal boundaries for netting and settlement. All FRAME objects are associated with exactly one window, and COLLAPSE objects are emitted at window close.

4.1 Window Definition

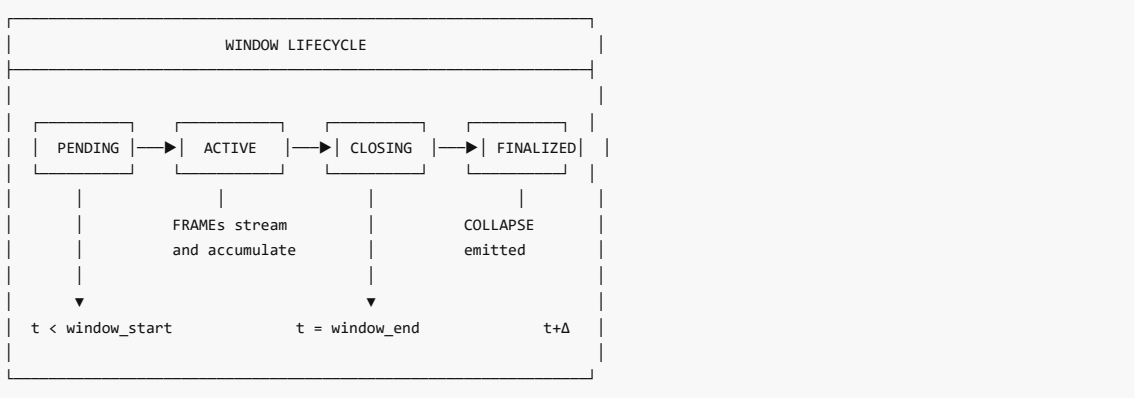
A clearing window is defined by:

1. **window_start**: The inclusive start time (ISO 8601)
2. **window_end**: The exclusive end time (ISO 8601)
3. **window_duration_ms**: Duration in milliseconds (MUST equal window_end - window_start)
4. **netting_algorithm**: The algorithm used to compute net values from frames

The `window_id` is computed as:

```
window_id = BLAKE3(canonical_encode({
  "window_start": <window_start>,
  "window_end": <window_end>,
  "window_duration_ms": <window_duration_ms>,
  "netting_algorithm": <netting_algorithm>
}))
```

4.2 Window Lifecycle



1. **PENDING**: Window is defined but not yet active. No frames accepted.
2. **ACTIVE**: Current time is within [window_start, window_end). Frames stream in.
3. **CLOSING**: Window has ended. No new frames accepted. Netting computation in progress.
4. **FINALIZED**: COLLAPSE emitted. Window is immutable.

4.3 Frame-Window Association

A FRAME is associated with a window if and only if:

1. The frame's `created_at` timestamp falls within [window_start, window_end)
2. All segment timeboxes fall within [window_start, window_end)

Frames that span multiple windows MUST be split into multiple frames, one per window.

4.4 Netting Process

At window close, the netting algorithm processes all valid frames:

1. Collect all frames with matching `intent_id` and valid window membership
2. Order frames by `seq_no`
3. For each segment in each frame, accumulate energy values
4. Apply loss estimates to compute net transfer
5. Compute confidence bounds from aggregate loss estimates
6. Emit COLLAPSE with net values

4.5 Standard Netting Algorithms

4.5.1 SIMPLE_SUM_V1

The simplest netting algorithm. For each intent:

```
net_energy_j = Σ (segment.energy_j_partial) - Σ (segment.loss_estimate.max)
net_energy_confidence.min = Σ (segment.energy_j_partial) - Σ (segment.loss_estimate.max)
net_energy_confidence.max = Σ (segment.energy_j_partial) - Σ (segment.loss_estimate.min)
```

4.5.2 WEIGHTED_CONFIDENCE_V1

Uses confidence-weighted averaging:

```
net_energy_j = Σ (segment.energy_j_partial * confidence_weight) / Σ (confidence_weight)
where confidence_weight = 1 / (segment.loss_estimate.max - segment.loss_estimate.min + 1)
```

4.6 Window Parameters

Implementations MUST support configurable window parameters:

Parameter	Type	Default	Description
`window_duration_ms`	uint64	3600000	Window duration (default: 1 hour)
`min_frames_for_collapse`	uint32	1	Minimum frames required to emit collapse
`collapse_delay_ms`	uint64	5000	Delay after window_end before collapse
`netting_algorithm`	string	"SIMPLE_SUM_V1"	Algorithm for netting computation

---

5. Canonical Encoding Rules

Deterministic encoding is critical for reproducible hashing and signature verification across implementations.

5.1 Interchange Format (JSON)

JSON is the primary format for human-readable interchange and debugging.

5.1.1 JSON Encoding Rules

1. **Character Encoding**: UTF-8, no BOM
2. **Numeric Values**:
   - Integers: No leading zeros, no plus sign
   - No floating point for energy values (use uint64 joules)
3. **String Values**:
   - Escape only required characters: `"`, `\`, and control characters
   - Use `\uXXXX` for control characters
4. **Binary Data**:
   - Encode as lowercase hexadecimal strings
   - No `0x` prefix
5. **Null Values**:
   - Use JSON `null`, never omit fields
6. **Arrays**:
   - No trailing commas
   - Preserve order (order is significant)
7. **Objects**:
   - No trailing commas
   - No duplicate keys

5.2 Wire Format (CBOR)

CBOR (RFC 8949) is the binary wire format for efficient transmission.

5.2.1 CBOR Encoding Rules

1. **Deterministic Encoding**: Use Core Deterministic Encoding (RFC 8949 Section 4.2)
2. **Map Key Ordering**: Lexicographic ordering of UTF-8 encoded keys
3. **Integer Encoding**: Smallest valid encoding
4. **Binary Data**: Major type 2 (byte string)
5. **Strings**: Major type 3 (text string), UTF-8
6. **Arrays**: Major type 4, preserve order
7. **Maps**: Major type 5, deterministic key ordering

5.3 Canonical Form

For hashing and signatures, objects MUST be encoded in canonical form.

5.3.1 Canonicalization Rules

1. **Field Ordering**: Alphabetical by field name (Unicode code point order)
2. **No Optional Fields**: All fields MUST be present. Use `null` for absent values.
3. **Computed Fields Excluded**: `intent_id`, `frame_id`, `collapse_id` are NOT included in canonical form (they are derived from it)
4. **Signature Exclusion**: `sigset` is NOT included in canonical form for that object
5. **Whitespace**: No whitespace in JSON canonical form
6. **Format**: CBOR for hashing, JSON for debugging only

5.3.2 Canonical Field Order

INTENT canonical fields (alphabetical):

```
created_at, crypto_suite_id, location_tag, max_energy_j, metadata,
modality_set, rx_node_pubkey, timebox, tx_node_pubkey, version
```

FRAME canonical fields (alphabetical):

```
created_at, crypto_suite_id, intent_id, metadata, segments, seq_no, version
```

COLLAPSE canonical fields (alphabetical):

```
created_at, crypto_suite_id, fees, finality_proof, intent_id, metadata,
net_energy_confidence, net_energy_j, version, window_id
```

5.4 Hash Computation

```
object_id = BLAKE3(cbor_canonical_encode(object_without_id_and_sigset))
```

The BLAKE3 hash produces a 32-byte (256-bit) digest.

---

6. Verification Rules

All verification rules are deterministic and MUST produce identical results across conforming implementations.

6.1 Signature Verification

6.1.1 Signature Structure

```
{
  "signer_pubkey": "<hex-encoded public key>",
  "signature": "<hex-encoded signature>",
  "crypto_suite_id": "<suite identifier>"
}
```

6.1.2 Verification Procedure

1. Extract `crypto_suite_id` from signature object
2. Validate that suite is in the supported registry
3. Compute canonical form of the object (excluding `sigset`)
4. Compute CBOR encoding of canonical form
5. Verify signature over CBOR bytes using suite-specific algorithm
6. Verify signer_pubkey is authorized for this object type

6.1.3 Authorization Rules

Object Type	Authorized Signers
INTENT	`tx_node_pubkey` OR `rx_node_pubkey`
FRAME	Any `from_pubkey` or `to_pubkey` in segments
COLLAPSE	Any member of `finality_proof.signer_set`

6.2 Timebox Validation

1. Parse `t_start` and `t_end` as ISO 8601 timestamps
2. Verify both are valid ISO 8601 with timezone (Z or offset)
3. Verify `t_start` < `t_end`
4. For INTENT activation: current_time >= `t_start` AND current_time < `t_end`

5. Verify timebox duration does not exceed implementation limits

6.3 Window Membership

A FRAME belongs to window W if and only if:

1. `frame.created_at >= W.window_start`
2. `frame.created_at < W.window_end`
3. For all segments S in frame:
   - `S.segment_timebox.t_start >= W.window_start`
   - `S.segment_timebox.t_end <= W.window_end`

6.4 Segment Consistency

For a FRAME to be segment-consistent:

1. **Ordering**: `segment_id` values MUST be sequential starting from 0
2. **Chaining**: For segments i and i+1: `segments[i].to_pubkey == segments[i+1].from_pubkey`
3. **Energy Conservation**:

   ```
   segments[i+1].energy_j_partial <= segments[i].energy_j_partial - segments[i].loss_estimate.min
   ```

4. **Temporal Ordering**:

   ```
   segments[i].segment_timebox.t_end <= segments[i+1].segment_timebox.t_start
   ```

5. **Modality Validity**: Each segment's modality MUST be in parent INTENT's `modality_set`

6.5 Energy Conservation Checks

6.5.1 Frame-Level Conservation

For each frame:

```
input_energy = segments[0].energy_j_partial
output_energy = segments[n-1].energy_j_partial
total_loss = Σ(loss_estimate.max for all segments)
REQUIRE: input_energy - output_energy <= total_loss
```

6.5.2 Intent-Level Conservation

Across all frames for an intent within a window:

```
total_input = Σ(first_segment.energy_j_partial for all frames)
total_output = Σ(last_segment.energy_j_partial for all frames)
REQUIRE: total_input >= total_output
REQUIRE: total_input <= intent.max_energy_j (SHOULD, not MUST)
```

6.6 Finality Proof Verification

For a COLLAPSE to have valid finality:

1. Recompute `window_id` from `window_params`; MUST match
2. Verify `merkle_root` is correctly computed from frame_ids
3. Verify threshold number of signatures in `finality_proof.signatures`
4. Each signature MUST:
   - Be from a pubkey in `signer_set`
   - Verify over canonical(merkle_root || window_id || net_energy_j)
5. No duplicate signers

---

7. Crypto-Agility

OERC-S is designed for cryptographic agility, supporting algorithm evolution without protocol changes.

7.1 Crypto Suite Registry

Each crypto suite is identified by a string and defines algorithms for:

- **Signing**: Digital signature algorithm
- **Verification**: Signature verification
- **Key Exchange**: (Optional) Key agreement protocol
- **Hashing**: Hash function for identifiers

7.2 Classical Suites

7.2.1 ED25519-BLAKE3

Component	Algorithm
Signing	Ed25519 (RFC 8032)
Verification	Ed25519
Key Exchange	X25519 (RFC 7748)
Hashing	BLAKE3
Public Key Size	32 bytes
Signature Size	64 bytes

Suite ID: `"ED25519-BLAKE3"`

7.2.2 ED25519-SHA256

Component	Algorithm
Signing	Ed25519 (RFC 8032)
Verification	Ed25519
Key Exchange	X25519 (RFC 7748)
Hashing	SHA-256
Public Key Size	32 bytes
Signature Size	64 bytes

Suite ID: `"ED25519-SHA256"`

7.3 Post-Quantum Suites

7.3.1 ML-DSA-65-BLAKE3

Component	Algorithm
Signing	ML-DSA-65 (FIPS 204)
Verification	ML-DSA-65
Key Exchange	ML-KEM-768 (FIPS 203)
Hashing	BLAKE3
Public Key Size	1952 bytes
Signature Size	3293 bytes

Suite ID: `"ML-DSA-65-BLAKE3"`

7.3.2 ML-DSA-87-BLAKE3

Component	Algorithm
Signing	ML-DSA-87 (FIPS 204)
Verification	ML-DSA-87
Key Exchange	ML-KEM-1024 (FIPS 203)
Hashing	BLAKE3
Public Key Size	2592 bytes
Signature Size	4595 bytes

Suite ID: `"ML-DSA-87-BLAKE3"`

7.4 Hybrid Suites

Hybrid suites combine classical and post-quantum algorithms for defense-in-depth.

7.4.1 ED25519-ML-DSA-65-BLAKE3

Component	Algorithm
Signing	Ed25519 + ML-DSA-65 (concatenated)
Verification	Both must verify
Key Exchange	X25519 + ML-KEM-768
Hashing	BLAKE3
Public Key Size	32 + 1952 = 1984 bytes
Signature Size	64 + 3293 = 3357 bytes

Suite ID: `"ED25519-ML-DSA-65-BLAKE3"`

Hybrid Signature Format:

```
{
  "classical": "<64 bytes Ed25519 signature, hex>",
  "pqc": "<3293 bytes ML-DSA-65 signature, hex>"
}
```

7.4.2 X25519-ML-KEM-768-BLAKE3

For key exchange only (used in encrypted transport, not object signing):

Component	Algorithm
Key Exchange	X25519 + ML-KEM-768
Hashing	BLAKE3
Shared Secret	BLAKE3(X25519_shared

Suite ID: `"X25519-ML-KEM-768-BLAKE3"`

7.5 Suite Negotiation

1. Objects MUST include `crypto_suite_id` field
2. Nodes SHOULD support multiple suites for interoperability
3. Nodes MUST support at least one classical suite (ED25519-BLAKE3)
4. Nodes operating in high-security contexts SHOULD support at least one hybrid suite
5. Suite deprecation follows registry-defined sunset policy

7.6 Suite Migration

When migrating from suite A to suite B:

1. During transition period, sign with both suites (dual sigset entries)
2. Verifiers accept either signature
3. After transition period, suite A signatures rejected
4. Old objects with suite A signatures remain valid (no retroactive invalidation)

---

8. Conformance Requirements

8.1 Conformance Levels

8.1.1 OERC-S Minimal

A **Minimal** implementation MUST:

- Parse and validate INTENT, FRAME, and COLLAPSE objects
- Compute object identifiers (intent_id, frame_id, collapse_id)
- Verify signatures for at least ED25519-BLAKE3 suite
- Implement canonical encoding (CBOR)
- Pass all Minimal test vectors

8.1.2 OERC-S Standard

A **Standard** implementation MUST:

- Meet all Minimal requirements
- Implement clearing window logic
- Support SIMPLE_SUM_V1 netting algorithm
- Generate valid COLLAPSE objects
- Support at least two crypto suites
- Pass all Standard test vectors

8.1.3 OERC-S Full

A **Full** implementation MUST:

- Meet all Standard requirements
- Support all registered crypto suites including hybrids
- Implement all registered netting algorithms
- Support configurable window parameters
- Pass all Full test vectors

8.2 Test Vectors

Conformance is verified through official test vectors. Test vectors are provided in JSON format and cover:

1. **Encoding Vectors**: Canonical form computation
2. **Hash Vectors**: Object ID computation
3. **Signature Vectors**: Sign and verify operations per suite
4. **Validation Vectors**: Valid and invalid object examples
5. **Netting Vectors**: Frame-to-collapse computation
6. **Window Vectors**: Window membership and lifecycle

8.3 Conformance Assertion

An implementation MAY assert "OERC-S Compliant" if and only if:

1. It passes 100% of required test vectors for its conformance level
2. It does not extend the protocol in incompatible ways
3. It publishes its conformance level and test results

8.4 Non-Conformance Consequences

No conformance => No liquidity.

Non-conforming implementations:

- MUST NOT assert OERC-S compliance
- SHOULD NOT expect interoperability with conforming implementations
- MAY be excluded from settlement networks
- MAY have their COLLAPSE objects rejected by clearinghouses

8.5 Interoperability Requirements

Conforming implementations MUST:

- Accept valid objects from any conforming implementation
- Produce objects that any conforming implementation can validate
- Not require proprietary extensions for basic operation
- Document any optional extensions clearly

---

9. Security Considerations

9.1 Threat Model

OERC-S assumes the following threat model:

1. **Network Adversary**: Attackers can observe, delay, replay, and modify messages
2. **Compromised Nodes**: Some nodes may be malicious or compromised
3. **Quantum Adversary** (future): Adversaries with quantum computers capable of breaking classical cryptography

9.2 Cryptographic Security

9.2.1 Hash Function Security

BLAKE3 provides:

- 256-bit pre-image resistance
- 256-bit second pre-image resistance
- 128-bit collision resistance

Implementations MUST NOT truncate hash outputs.

9.2.2 Signature Security

All signature schemes provide existential unforgeability under chosen message attack (EUF-CMA):

- Ed25519: ~128-bit classical security
- ML-DSA-65: NIST Level 3 security (~143-bit classical, ~128-bit quantum)
- Hybrid: Security of stronger algorithm

9.2.3 Key Management

Implementations MUST:

- Generate keys using cryptographically secure random number generators
- Protect private keys with appropriate access controls
- Support key rotation without protocol changes
- Maintain key-to-identity binding through external PKI

9.3 Replay Protection

9.3.1 Object-Level Replay

Object identifiers (intent_id, frame_id, collapse_id) provide natural replay protection:

- Identical objects have identical IDs
- Implementations MUST reject duplicate IDs within scope

9.3.2 Intent Replay

INTENTs include timeboxes that limit validity windows. Implementations MUST:

- Reject INTENTs with `t_end` in the past
- Track active intent_ids to prevent reactivation

9.3.3 Frame Replay

Frames include sequence numbers. Implementations MUST:

- Reject frames with seq_no <= highest seen for that intent
- Reject frames for expired or inactive intents

9.4 Denial of Service

9.4.1 Resource Exhaustion

Implementations SHOULD:

- Limit maximum object sizes
- Rate-limit object acceptance per source
- Validate signatures before expensive operations
- Implement backpressure mechanisms

9.4.2 Computational DoS

Signature verification is computationally expensive. Implementations SHOULD:

- Cache verification results
- Prioritize objects from known nodes
- Implement proof-of-work or stake-based prioritization

9.5 Privacy Considerations

9.5.1 Data Exposure

OERC-S objects contain:

- Public keys (node identifiers)
- Energy quantities
- Timing information
- Location tags

This data may reveal:

- Energy trading patterns
- Orbital positions
- Business relationships

9.5.2 Privacy Mitigations

Implementations MAY:

- Use ephemeral keys for unlinkability

- Aggregate frames before broadcast
- Encrypt objects at transport layer
- Use private clearinghouses

9.6 Quantum Resistance

9.6.1 Timeline

Crypto suites are registry-driven; deployments choose migration policy. OERC-S provides:

- Hybrid suites for immediate deployment
- Pure PQC suites for future migration
- Crypto-agility for algorithm updates

9.6.2 Migration Strategy

1. Deploy hybrid suites when post-quantum readiness is required
2. Mandate hybrid suites per deployment policy
3. Deprecate classical suites when quantum threat materializes
4. Evaluate pure PQC deployment based on ecosystem readiness

---

10. IANA Considerations

This section is a placeholder for future IANA registrations.

10.1 Media Types

The following media types are requested for registration:

Media Type	Description
`application/oerc-s+json`	OERC-S object in JSON format
`application/oerc-s+cbor`	OERC-S object in CBOR format

10.2 URI Schemes

The following URI scheme is requested:

Scheme	Description
`oerc-s:`	OERC-S object identifier

Format: `oerc-s:<object_type>/<object_id_hex>`

Examples:

- `oerc-s:intent/c3d4e5f6a7b8c9d0e1f2a3b4c5d6e7f8a9b0c1d2e3f4a5b6c7d8e9f0a1b2c3d4`
- `oerc-s:frame/1234567890abcdef1234567890abcdef1234567890abcdef1234567890abcdef`
- `oerc-s:collapse/abcdef0123456789abcdef0123456789abcdef0123456789abcdef0123456789`

10.3 Crypto Suite Registry

A registry of OERC-S crypto suites should be established with the following initial entries:

Suite ID	Status	Reference
ED25519-BLAKE3	Active	This document
ED25519-SHA256	Active	This document
ML-DSA-65-BLAKE3	Active	This document
ML-DSA-87-BLAKE3	Active	This document
ED25519-ML-DSA-65-BLAKE3	Active	This document

10.4 Netting Algorithm Registry

A registry of OERC-S netting algorithms should be established:

Algorithm ID	Status	Reference
SIMPLE_SUM_V1	Active	This document

WEIGHTED_CONFIDENCE_V1	Active	This document

Appendix A: Test Vectors

A.1 Canonical Encoding Vector

Input INTENT (JSON, non-canonical):


```
{
  "version": "OERC-S/0.1",
  "tx_node_pubkey": "0102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f20",
  "rx_node_pubkey": "2120191817161514131211100f0e0d0c0b0a090807060504030201000fffefdfb",
  "timebox": {
    "t_start": "2025-01-01T00:00:00Z",
    "t_end": "2025-01-01T01:00:00Z"
  },
  "max_energy_j": 1000000000,
  "modality_set": ["laser"],
  "location_tag": {
    "orbit_regime": "LEO",
    "ephemeris_hash": "0000000000000000000000000000000000000000000000000000000000000000",
    "footprint_id": "TEST-001"
  },
  "crypto_suite_id": "ED25519-BLAKE3",
  "sigset": [],
  "created_at": "2024-12-31T23:59:00Z",
  "metadata": null
}
```


Canonical CBOR (hex):


```
a96a6372656174656 45f617474323032342d31322d33315432333a35393a30305a
[See conformance/vectors/ for full encoding]
```


(Full vector available in official test suite)

Computed intent_id:


```
e3b0c44298fc1c149afbf4c8996fb92427ae41e4649b934ca495991b7852b855
```


(Computed by conformance runner from canonical encoding)

A.2 Signature Vector

Message (canonical CBOR, hex):


```
[See conformance/vectors/ for full vector]
```


Test-only deterministic key material (Ed25519, hex):


```
9d61b19deffd5a60ba844af492ec2cc44449c5697b326919703bac031cae7f60
```


Public Key (Ed25519, hex):


```
d75a980182b10ab7d54bfed3c964073a0ee172f3daa62325af021a68f707511a
```


Signature (hex):


```
e5564300c360ac729086e2cc806e828a84877f1eb8e5d974d873e065224901555fb8821590a33bacc61e39701cf9b46bd25bf5f0595bbe24655141438e7a100b
```

Appendix B: JSON Schema

B.1 INTENT Schema


```
{
  "$schema": "https://json-schema.org/draft/2020-12/schema",
  "$id": "urn:oerc-s:schema:v0.1:intent",
```

```
  "title": "OERC-S Intent",
  "type": "object",
  "required": [
    "version", "tx_node_pubkey", "rx_node_pubkey", "timebox",
    "max_energy_j", "modality_set", "location_tag", "crypto_suite_id",
    "sigset", "created_at", "metadata"
  ],
  "properties": {
    "version": {
      "type": "string",
      "const": "OERC-S/0.1"
    },
    "tx_node_pubkey": {
      "type": "string",
      "pattern": "^[0-9a-f]{64}$"
    },
    "rx_node_pubkey": {
      "type": "string",
      "pattern": "^[0-9a-f]{64}$"
    },
    "timebox": {
      "type": "object",
      "required": ["t_start", "t_end"],
      "properties": {
        "t_start": {"type": "string", "format": "date-time"},
        "t_end": {"type": "string", "format": "date-time"}
      }
    },
    "max_energy_j": {
      "type": "integer",
      "minimum": 1
    },
    "modality_set": {
      "type": "array",
      "items": {
        "type": "string",
        "enum": ["laser", "microwave", "storage", "relay"]
      },
      "minItems": 1
    },
    "location_tag": {
      "type": "object",
      "required": ["orbit_regime", "ephemeris_hash", "footprint_id"],
      "properties": {
        "orbit_regime": {"type": "string"},
        "ephemeris_hash": {"type": "string", "pattern": "^[0-9a-f]{64}$"},
        "footprint_id": {"type": ["string", "null"]}
      }
    },
    "crypto_suite_id": {"type": "string"},
    "sigset": {
      "type": "array",
      "items": {"$ref": "#/$defs/signature"}
    },
    "created_at": {"type": "string", "format": "date-time"},
    "metadata": {"type": ["object", "null"]}
  },
  "$defs": {
    "signature": {
      "type": "object",
      "required": ["signer_pubkey", "signature", "crypto_suite_id"],
      "properties": {
        "signer_pubkey": {"type": "string"},
        "signature": {"type": "string"},
        "crypto_suite_id": {"type": "string"}
      }
    }
  }
}
```

Appendix C: Minimal Proof to Publish (One real test)

A partner organization can demonstrate OERC-S conformance by publishing the following fields from ONE real Collapse:

Field	Description
`collapse_id`	BLAKE3 hash identifying the collapse
`window_id`	Clearing window identifier
`net_energy_j`	Net energy transferred (joules)
`finality_proof.merkle_root`	Merkle root of frames in window
`signer_set` (optional)	Public keys of finality attesters

Publishing these fields enables third parties to reference and verify the test without revealing proprietary telemetry or operational details.

CTA: Emit ONE real Collapse hash from a test environment.

References

Normative References

- **RFC 2119**: Key words for use in RFCs to Indicate Requirement Levels
- **RFC 8032**: Edwards-Curve Digital Signature Algorithm (EdDSA)
- **RFC 7748**: Elliptic Curves for Security (X25519)
- **RFC 8949**: Concise Binary Object Representation (CBOR)
- **RFC 3339**: Date and Time on the Internet: Timestamps
- **FIPS 203**: Module-Lattice-Based Key-Encapsulation Mechanism (ML-KEM)
- **FIPS 204**: Module-Lattice-Based Digital Signature Algorithm (ML-DSA)
- **BLAKE3**: BLAKE3 Cryptographic Hash Function Specification

Informative References

- **RFC 8610**: Concise Data Definition Language (CDDL)
- **ISO 8601**: Date and time format
- **NIST SP 800-208**: Recommendation for Stateful Hash-Based Signature Schemes

---

Document History

Version	Date	Changes
0.1	2025-12-27	Initial draft specification

End of OERC-S v0.1 Specification