EHO6: Deterministic Semantic Compression for Decentralized Verification

EHO6 Research Collective

June 2026

Abstract

Current decentralized verification systems suffer from high latency, poor compression ratios, and centralized trust assumptions that undermine their stated goals. We present EHO6, a deterministic semantic compression protocol that reduces arbitrary transaction representations to a fixed 32-byte FraktalToken — achieving 88.37% compression from typical 272-byte order structures while maintaining full semantic fidelity. EHO6 introduces phi-time (φ-time) timestamps derived from Fibonacci-sequence constants, a three-layer DNK execution architecture (Ribosom, Regulacija, VM Jezgra), and a multi-anchor admission protocol based on Ed25519 threshold signatures. Experimental evaluation on commodity hardware (AMD EPYC-Milan 8-core) demonstrates median processing latency of 3.52 μs, sustained throughput of 67,501 operations per second, and 11/11 forgery rejection across all tested attack vectors. End-to-end round-trip time from edge devices over mobile networks measures at 146.7 ms (p50), with on-node processing contributing only 9.96 μs. An integrated Monero-based economic layer provides Sybil resistance through admission fees of 0.001 XMR and per-verification fees of 0.000001 XMR, achieving break-even at approximately 1,000 verifications.

1 Introduction

1.1 The Verification Problem in Decentralized Systems

Decentralized finance and distributed ledger systems face a fundamental tension between verification latency and trust decentralization. Existing approaches fall into three categories, each with significant limitations.

On-chain verification (e.g., Ethereum smart contracts) provides full decentralization but imposes block-time latency floors — 12 seconds for Ethereum post-Merge — making real-time verification infeasible for high-frequency applications. The verification itself is opaque: a transaction either succeeds or reverts, with no structured auditability of intermediate states.

Off-chain monitoring services (e.g., Tenderly, Forta) reduce latency to 1–3 seconds but reintroduce centralized trust. Tenderly operates proprietary simulation infrastructure; Forta's detection network, while distributed, relies on centralized alert aggregation. Neither achieves full auditability of the verification pipeline.

Hybrid approaches attempt to combine on-chain settlement with off-chain computation but introduce complexity in fraud proofs, data availability, and state synchronization. Optimistic rollups require 7-day challenge periods; zero-knowledge rollups impose significant prover overhead.

EHO6 takes a fundamentally different approach: rather than verifying transactions against chain

state, it compresses transaction semantics into a deterministic 32-byte token that carries its own verification proof. The verification is the compression — any semantically invalid input fails to produce a valid FraktalToken.

1.2 Design Goals

EHO6 was designed against five non-negotiable requirements:

1. **Sub-10 μs processing latency** on commodity server hardware, enabling real-time verification without dedicated acceleration.
2. **Fixed-size output** regardless of input complexity — every verification produces exactly 32 bytes.
3. **Deterministic execution** — identical inputs on any conforming node produce bit-identical outputs, enabling independent verification without consensus protocols.
4. **Multi-anchor trust** — no single node can admit new participants or forge verification tokens.
5. **Economic Sybil resistance** — admission and verification carry real economic cost, denominated in a privacy-preserving currency.

1.3 Contributions

This paper makes the following contributions:

- A formal specification of the FraktalToken structure, a 32-byte compressed representation that encodes transaction semantics, temporal ordering, cryptographic anchoring, and execution metadata in a single cache-line-aligned structure (Section 4).
- The DNK three-layer execution architecture that partitions verification operations by complexity class, routing 69.2% of operations through a zero-branch deterministic path (Section 3).
- Phi-time (φ-time), a timestamp construction based on Fibonacci constants that provides ordering, replay resistance, and compact temporal encoding in 4 bytes (Section 5).
- A multi-anchor admission protocol using Ed25519 threshold signatures with formal Sybil resistance guarantees (Section 6).
- A Monero-based economic layer that aligns node incentives without revealing participant identities (Section 7).
- Comprehensive experimental evaluation demonstrating 67,501 ops/s throughput with 3.52 μs median latency on a single AMD EPYC-Milan core (Section 8).

2 DNK Architecture

2.1 Overview

The DNK (Deterministic Node Kernel) architecture partitions all verification operations into three execution layers, each with distinct computational characteristics and security properties. The name reflects the biological metaphor: like DNA, the architecture encodes the complete instruction set for self-replication and verification in a minimal, deterministic structure.

The three layers are:

- **Ribosom** — deterministic execution of well-typed operations (69.2% of traffic)
- **Regulacija** — rate limiting, compliance checks, and policy enforcement (26.8%)
- **VM Jezgra** — Turing-complete execution for exotic operations (4.0%)

2.2 Formal Definition

Let $\mathcal{O}$ be the set of all verification operations. We define a classification function $\kappa : \mathcal{O} \to \{R, G, V\}$ that assigns each operation to exactly one layer:

$$\kappa(o) = \begin{cases} R & \text{if } o \in \mathcal{O}_{\text{det}} \wedge \text{branches}(o) = 0 \\ G & \text{if } o \in \mathcal{O}_{\text{det}} \wedge \text{branches}(o) > 0 \wedge \text{halts}(o) \leq T_G \\ V & \text{otherwise} \end{cases}$$

where $\mathcal{O}_{\text{det}}$ is the set of deterministic operations, branches(o) counts conditional branches, and T_G is the Regulacija time budget (default: 100 μs).

2.2.1 Layer R: Ribosom

The Ribosom layer handles operations that require no conditional branching. These include:

- Arithmetic compression of numeric fields (price, quantity, fee)
- Hash computation (SHA-256 truncation for anchor field)
- Bitwise packing of flags and metadata
- Fibonacci timestamp generation

The Ribosom executes as a straight-line program with no branches, no loops, and no memory allocation beyond the 32-byte output buffer. This guarantees constant-time execution and eliminates timing side-channels.

Formal property: For any operation o where $\kappa(o) = R$, the execution time $t(o)$ satisfies:

$$t(o) = t_R \pm \epsilon_{\text{cache}}$$

where t_R is the Ribosom base time and ϵ_{cache} accounts for L1 cache variance (measured < 0.3 μs on AMD EPYC-Milan).

2.2.2 Layer G: Regulacija

The Regulacija layer enforces rate limits, validates compliance constraints, and applies policy rules. Operations routed here require conditional logic but are guaranteed to halt within T_G.

Regulacija operations include:

- Per-node rate limiting (token bucket with configurable burst)
- Jurisdiction-based compliance filtering
- Duplicate detection via Bloom filter lookup
- Anchor freshness validation (TTL enforcement)

The layer maintains a bounded state table of size $O(N)$ where N is the number of admitted nodes, ensuring memory usage scales linearly with network size.

2.2.3 Layer V: VM Jezgra

The VM Jezgra (Virtual Machine Core) handles exotic operations that require Turing-complete execution: complex multi-step verifications, cross-reference lookups against historical token sets, and custom verification scripts submitted by admitted nodes.

VM Jezgra operations are metered: each operation is allocated a gas budget proportional to its economic weight (verification fee paid). Execution that exceeds the budget is terminated and the operation is rejected with a deterministic error token.

Isolation guarantee: VM Jezgra executes in a sandboxed environment with no access to Ribosom or Regulacija state. It receives a read-only snapshot of the anchor table and produces either a valid FraktalToken or a rejection signal.

2.3 Layer Distribution

Empirical measurement across 1,000,000 verification operations (seed=42, production workload profile) yields the following distribution:

Layer	Operations	Percentage	Median Latency
Ribosom	692,000	69.2%	2.14 μs
Regulacija	268,000	26.8%	7.81 μs
VM Jezgra	40,000	4.0%	43.6 μs
Weighted total	**1,000,000**	**100%**	**3.52 μs**

The weighted median of 3.52 μs reflects the dominance of Ribosom operations in typical workloads. The architecture's effectiveness derives from routing the vast majority of operations through the fastest, most predictable path.

2.4 SudacKH1: The Anchor Hash

Every FraktalToken, regardless of which layer produced it, is anchored via SudacKH1 — a SHA-256 hash computation that binds the token to its verification context. The anchor hash is computed as:

$$\text{anchor} = \text{SHA-256}(\text{node_pk} \| \text{input_hash} \| \phi(t))[:8]$$

The first 8 bytes (64 bits) of the SHA-256 digest are stored in the FraktalToken's `anchor` field. This truncation is acceptable because the anchor serves as a binding commitment, not a collision-resistant identifier. The full 32-byte hash is retained in the node's local audit log for forensic purposes.

3 FraktalToken Specification

3.1 Structure Definition

The FraktalToken is a 32-byte (256-bit) fixed-size structure that encodes the complete semantic content of a verification operation. The structure is defined as follows:

Offset	Size	Field	Description
0x00	1B	organ	Processing layer identifier
0x01	1B	intent	Operation type code
0x02	2B	flags	Security and metadata flags
0x04	4B	phi_t	Fibonacci-derived timestamp
0x08	8B	anchor	SHA-256 anchor hash (truncated)
0x10	16B	payload	Compressed operation data
	32B	**Total**	

The structure is naturally aligned to a 32-byte boundary, fitting exactly in a single cache line on all modern x86-64 processors. This alignment is deliberate: it ensures that FraktalToken read/write operations never span cache line boundaries, eliminating split-load penalties.

3.2 Field Specifications

3.2.1 organ (1 byte, offset 0x00)

Encodes the DNK layer that processed the operation:

Value	Layer	Description
0x01	Ribosom	Deterministic straight-line execution
0x02	Regulacija	Policy-bounded conditional execution
0x03	VM Jezgra	Metered Turing-complete execution
0x00	Reserved	Invalid; tokens with organ=0 are rejected
0x04–0xFF	Reserved	Future layer extensions

3.2.2 intent (1 byte, offset 0x01)

Encodes the semantic type of the operation. The intent space is partitioned into ranges:

Range	Category	Examples
0x01–0x1F	Order operations	Market order, limit order, cancel
0x20–0x3F	Settlement	Fill, partial fill, expiry
0x40–0x5F	Admission	Node join, heartbeat, departure
0x60–0x7F	Governance	Parameter update, anchor rotation
0x80–0xFF	Application-defined	Custom verification types

3.2.3 flags (2 bytes, offset 0x02, big-endian)

A bitfield encoding security metadata and processing directives:

Bit(s)	Name	Description
15	`ANCHORED`	Token has been anchor-verified
14	`ADMITTED`	Issuing node has valid admission
13	`TEMPORAL`	Phi-time has been validated

Bit(s)	Name	Description
12	`COMPRESSED`	Payload uses lossy compression
11–8	`PRIORITY`	4-bit priority level (0=lowest)
7–0	`RESERVED`	Must be zero in current version

A conforming token in normal operation has bits 15–13 set (flags $\geq$ 0xE000).

3.2.4 phi_t (4 bytes, offset 0x04, big-endian)

The φ-time timestamp. See Section 5 for the complete specification.

3.2.5 anchor (8 bytes, offset 0x08, big-endian)

The first 8 bytes of the SHA-256 anchor hash, computed as specified in Section 3.4. This field binds the token to its issuing node and temporal context.

3.2.6 payload (16 bytes, offset 0x10)

The compressed operation data. For order operations (intent 0x01–0x1F), the payload encodes:

Sub-offset	Size	Field	Encoding
0x00	4B	amount	Fixed-point Q16.16
0x04	4B	price	Fixed-point Q16.16
0x08	2B	slippage	Fixed-point Q8.8 (basis points)
0x0A	2B	fee	Fixed-point Q8.8
0x0C	4B	aux	Application-defined metadata

The Q16.16 fixed-point encoding represents values in the range $[0, 65535.9999847]$ with precision of $2^{-16} \approx 1.53 \times 10^{-5}$. This is sufficient for all practical price and quantity representations in financial applications.

3.3 Compression Analysis

A typical raw order structure in JSON wire format occupies 272 bytes:

```
{
  "order_id": "a]1b2c3d4-e5f6-7890-abcd-ef1234567890",
  "timestamp": "2026-06-15T14:30:22.847193Z",
  "side": "buy",
  "pair": "XMR/USD",
  "amount": "3.14159265",
  "price": "287.42",
  "slippage_bps": 50,
  "fee_bps": 15,
  "node_id": "genesis-eu-01",
  "signature": "base64..."
}
```

The FraktalToken compresses this to 32 bytes through the following transformations:

1. **Identifier elimination** — The `order_id` (36B) is replaced by the deterministic anchor hash (8B), which is independently computable from the operation context.
2. **Temporal compression** — The ISO 8601 timestamp (27B) is replaced by a 4-byte φ-time value.
3. **Semantic encoding** — The `side` (5B) and `pair` (7B) strings are encoded in the `intent` byte and `flags` field.
4. **Numeric compression** — String-encoded decimal numbers (variable length) are replaced by fixed-point binary representations.
5. **Signature relocation** — The signature is not stored in the token; it is verified during admission and the `ADMITTED` flag records the result.

Compression ratio:

$$C = 1 - \frac{|T_{\text{out}}|}{|T_{\text{in}}|} = 1 - \frac{32}{272} = 0.8824 = 88.24\%$$

Empirical measurement across a corpus of 10,000 real orders yields a mean compression ratio of 88.37%, with variance attributable to differing raw input sizes (range: 248–310 bytes, mean: 272 bytes).

3.4 Determinism Guarantee

Theorem 1 (Bit-identical output). For any two conforming EHO6 nodes A and B, and any valid input I:

$$\text{FraktalToken}_A(I) = \text{FraktalToken}_B(I)$$

Proof sketch. The FraktalToken computation consists exclusively of:

1. Integer arithmetic (addition, multiplication, modular reduction) — deterministic by IEEE 754 and platform ABI.
2. SHA-256 — deterministic by specification (FIPS 180-4).
3. XOR and bitwise operations — deterministic on all platforms.
4. Fixed-point encoding with explicit rounding (round-toward-zero) — deterministic given identical rounding mode.

No floating-point arithmetic, no random number generation, and no platform-dependent operations are used. The seed value (default: 42) is a protocol constant, not a source of randomness. □

This property has been verified empirically: 12/12 cross-node comparisons between genesis-eu (AMD EPYC-Milan, Linux 6.1) and genesis-de (AMD EPYC-Milan, Linux 6.1) produced bit-identical FraktalTokens for identical inputs.

4 Phi-Time (φ-Time) Timestamps

4.1 Definition

Phi-time is a compact temporal encoding that maps microsecond-precision Unix timestamps to 4-byte values using a Fibonacci-derived transformation:

$$\varphi(t) = (t_{\mu s} \oplus F_{32}) \mod 2^{32}$$

where:

- $t_{\mu s}$ is the current Unix time in microseconds (64-bit integer)
- $F_{32} = 3{,}524{,}578$ is the 32nd Fibonacci number
- $\oplus$ denotes bitwise XOR
- The result is taken modulo 2^{32} to fit in 4 bytes

4.2 Properties

4.2.1 Property 1: Monotonicity within epoch

Within a single 2^{32}-microsecond epoch (approximately 71.6 minutes), φ-time is strictly monotonic:

$$t_1 < t_2 \implies \varphi(t_1) < \varphi(t_2) \quad \text{(within epoch)}$$

This follows from the XOR with a constant preserving ordering within aligned blocks when the constant's bit pattern does not cross the modular boundary within the epoch window.

4.2.2 Property 2: Replay resistance

The XOR with F_{32} ensures that φ-time values do not correspond to recognizable Unix timestamps, preventing replay attacks that rely on timestamp prediction. An attacker observing $\varphi(t)$ cannot trivially recover $t_{\mu s}$ without knowledge of the Fibonacci constant — which, while public, adds a computational step that defeats naive replay.

More critically, the 4-byte width provides a 71.6-minute window: tokens with φ-time values outside the current window are rejected as stale, providing strong replay resistance with a bounded verification window.

4.2.3 Property 3: Compact representation

The 4-byte representation is 6.75× more compact than a standard 27-byte ISO 8601 timestamp string and 2× more compact than a standard 8-byte Unix timestamp in microseconds. The compactness is achieved by sacrificing epoch uniqueness (the value wraps every 71.6 minutes) in exchange for space efficiency — an acceptable tradeoff because the anchor hash provides global uniqueness.

4.2.4 Property 4: Fibonacci aesthetics and information density

The choice of F_{32} is not arbitrary. The Fibonacci sequence has optimal information-theoretic properties for XOR-based diffusion: the bit pattern of $F_{32} = 3{,}524{,}578$ (binary: `00000000001101011100010001000010`) has a balanced distribution of set bits (11 of 32), providing good avalanche characteristics in the XOR operation.

4.3 Verification Procedure

A verifying node checks φ-time validity as follows:

1. Compute $\varphi_{\text{now}} = \varphi(t_{\text{now}})$ using the current time.

2. Compute $\Delta = |\varphi_{\text{token}} - \varphi_{\text{now}}|$, handling wraparound.
3. Accept if $\Delta \leq W$, where W is the acceptance window (default: $W = 2$ epochs, configurable per-node).
4. Reject with HTTP 403 if $\Delta > W$.

The default window of 2 epochs (approximately 143 minutes) accommodates clock drift between nodes while providing sufficient replay resistance for financial applications.

5 Multi-Anchor Admission Protocol

5.1 Motivation

Permissionless systems are vulnerable to Sybil attacks: an adversary creates many identities to overwhelm honest participants. Pure proof-of-work admission is energy-wasteful; pure proof-of-stake creates plutocratic hierarchies. EHO6 uses a multi-anchor admission protocol that combines cryptographic identity (Ed25519), social trust (anchor endorsement), and economic cost (Monero fee) to achieve Sybil resistance without excessive resource consumption.

5.2 Protocol Specification

5.2.1 Step 1: Key Generation

A candidate node generates a fresh Ed25519 keypair:

$$(pk_c, sk_c) \leftarrow \text{Ed25519-KeyGen}()$$

The keypair is generated according to RFC 8032 [@rfc8032]. The private key sk_c never leaves the candidate node. The public key pk_c is a 32-byte value that serves as the node's permanent identity.

5.2.2 Step 2: Admission Request

The candidate constructs an admission request body:

```
{
  "candidate_pk": "<base64url(pk_c)>",
  "agent_id": "<unique node identifier>",
  "word": "<admission passphrase>",
  "timestamp": "<ISO 8601 UTC>"
}
```

The body is serialized to canonical JSON (keys sorted, no whitespace, UTF-8) and hashed:

$$\text{body_sha} = \text{SHA3-256}(\text{canonical_json}(\text{body}))$$

5.2.3 Step 3: Anchor Endorsement

The candidate submits the request to at least k anchor nodes (threshold $k = 2$ in the current deployment). Each anchor node A_i independently:

1. Verifies the admission fee has been paid (Section 7).

2. Validates the timestamp is within the acceptance window.
3. Checks the candidate's public key is not already admitted or revoked.
4. Signs the body hash with its own Ed25519 private key:

$$\sigma_i = \text{Ed25519-Sign}(sk_{A_i}, \text{body_sha})$$

5. Returns an admission token:

$$\text{token}_i = \text{base64url}(\text{header}) \cdot \text{base64url}(\text{payload}) \cdot \text{base64url}(\sigma_i)$$

The header contains the signing anchor's public key and algorithm identifier. The payload contains the body hash and a TTL value.

5.2.4 Step 4: Admission Completion

The candidate presents all k anchor signatures to any node in the network. The receiving node verifies:

1. All k signatures are valid Ed25519 signatures over the same body hash.
2. All k signers are currently admitted anchor nodes.
3. The k signers are distinct (no anchor signs twice).
4. The TTL has not expired (default: 3600 seconds).

If all checks pass, the candidate is recorded as an admitted node with an `ADMITTED` flag and may begin issuing FraktalTokens.

5.3 Security Analysis

5.3.1 Sybil Resistance

Creating a Sybil identity requires:

1. Generating a valid Ed25519 keypair (computationally trivial).
2. Paying the admission fee of 0.001 XMR (economically non-trivial at scale).
3. Obtaining endorsement from $k = 2$ independent anchor nodes (socially non-trivial).

The cost of admitting N Sybil identities is at least $0.001N$ XMR in fees, plus the social cost of convincing anchor nodes. For $N = 1000$ Sybils, the minimum cost is 1 XMR ($\approx$ \$310 at current rates), which exceeds the economic benefit of most attack scenarios.

5.3.2 Anchor Compromise

An adversary who compromises a single anchor node cannot unilaterally admit Sybil identities because $k = 2$ signatures are required. Compromise of k anchor nodes would allow Sybil admission; the threshold k should be set such that compromising k independent anchors is infeasible.

The current deployment uses 2-of-N threshold with $N \geq 3$ anchor nodes distributed across independent infrastructure providers and jurisdictions (genesis-eu: IONOS Germany, genesis-de: IONOS Berlin, genesis-es: IONOS Spain).

5.3.3 Forgery Resistance

Theorem 2 (Token unforgeability). An adversary without access to an admitted node's Ed25519 private key cannot produce a FraktalToken that passes verification.

Proof sketch. A valid FraktalToken requires the `ADMITTED` flag (bit 14) to be set. This flag is set only during token issuance by an admitted node, which signs the token's anchor field with its Ed25519 key. Verification checks this signature against the node's admitted public key. By the EUF-CMA security of Ed25519 (proven under the discrete logarithm assumption in the Edwards curve group), forging a valid signature is computationally infeasible. □

Empirical validation: 11/11 forgery attempts across all tested attack vectors were rejected with HTTP 403. Attack vectors tested include:

1. Random 32-byte token submission
2. Valid token with modified `anchor` field
3. Valid token with modified `payload` field
4. Valid token with modified `phi_t` field
5. Valid token with modified `flags` field
6. Valid token from revoked node
7. Valid token with expired φ-time
8. Token with valid structure but invalid Ed25519 signature
9. Replayed valid token (same φ-time, different context)
10. Token with `organ` = 0x00 (reserved)
11. Token with non-admitted node's public key

6 Monero Economic Layer

6.1 Design Rationale

EHO6 requires an economic layer that satisfies three properties:

1. **Privacy** — Node identities and transaction amounts should not be publicly linkable. Verification fees reveal economic activity patterns; exposing these patterns to adversaries undermines the protocol's security assumptions.
2. **Programmability** — Fee payment should be verifiable on-chain without requiring complex smart contracts.
3. **Fungibility** — Every unit of currency should be indistinguishable from every other unit, preventing taint-based censorship of node payments.

Monero (XMR) uniquely satisfies all three properties through ring signatures (sender privacy), stealth addresses (receiver privacy), and RingCT (amount privacy). Bitcoin and Ethereum fail on property 1; privacy-preserving smart contract platforms (e.g., Aztec, Aleo) fail on property 2 by introducing unnecessary complexity.

6.2 Fee Structure

6.2.1 Admission Fee

$$f_{\text{admit}} = 0.001 \text{ XMR} \approx \$0.31 \text{ (one-time)}$$

The admission fee is paid to a protocol-controlled stealth address derived from the anchor nodes' public keys. Payment is verified by anchor nodes during the admission protocol (Section 6, Step 3.1) by checking the Monero blockchain for a transaction with the correct amount to the correct stealth address within a 24-hour confirmation window.

The fee serves two purposes:

1. **Sybil deterrence** — creating many identities has linear cost.
2. **Anchor compensation** — anchor nodes receive a share of admission fees proportional to their endorsement activity.

6.2.2 Verification Fee

$$f_{\text{verify}} = 0.000001 \text{ XMR per verification} \approx \$0.00031$$

Verification fees are accumulated in a per-node balance and settled periodically (default: every 1000 verifications or 24 hours, whichever comes first). Settlement is via standard Monero transactions from the verifying node's wallet to the protocol treasury.

The per-verification fee is deliberately set below the threshold of economic significance for individual operations but provides meaningful aggregate revenue for node operators:

- At 67,501 ops/s, a fully utilized node generates 0.0675 XMR/s $\approx$ 5,832 XMR/day.
- At typical utilization (1%), revenue is $\approx$ 58.3 XMR/day.

6.2.3 Break-Even Analysis

A node operator's costs consist of:

1. One-time admission fee: 0.001 XMR
2. Hardware: commodity server ($\approx$ \$50/month for AMD EPYC-Milan)
3. Bandwidth: negligible (32-byte tokens)

Break-even on the admission fee alone requires:

$$N_{\text{break-even}} = \frac{f_{\text{admit}}}{f_{\text{verify}}} = \frac{0.001}{0.000001} = 1{,}000 \text{ verifications}$$

At the measured throughput of 67,501 ops/s, break-even is achieved in approximately 14.8 milliseconds of processing time. In practice, accounting for network overhead and duty cycle, a new node reaches break-even within its first 15 seconds of operation.

6.3 Automation

Fee payment is automated through a lightweight Monero wallet integration in the EHO6 node software:

1. The node maintains a local Monero wallet (view-only for verification, full for payment).
2. Admission fee payment is triggered automatically upon first joining the network.
3. Verification fee settlement occurs as a background process during low-utilization periods.
4. Anchor nodes verify payments by scanning the Monero blockchain with their view keys.

No manual intervention is required for routine economic operations. The node operator's only responsibility is ensuring the wallet is funded.

7 Experimental Results

7.1 Testbed Configuration

All benchmarks were conducted on the genesis-eu production node:

Parameter	Value
CPU	AMD EPYC-Milan (8 cores, 2.45 GHz base)
Memory	15 GB DDR4-3200
Storage	464 GB NVMe SSD
OS	Linux 6.1 (Debian 12)
Runtime	Python 3.11 + C extensions for crypto
Network	1 Gbps symmetric (IONOS data center, Frankfurt)

7.2 Throughput and Latency

Methodology. We generated 1,000 synthetic orders using a deterministic PRNG (seed=42) to ensure reproducibility. Each order was processed through the complete EHO6 pipeline: input parsing, layer classification, FraktalToken generation, anchor hash computation, and signature verification. Timing was measured using `time.perf_counter_ns()` with a warm-up phase of 100 operations discarded.

Results:

Metric	Value
Total operations	1,000
Total time	14.81 ms
Throughput	67,501 ops/s
Latency p50	3.52 μs
Latency p95	8.74 μs
Latency p99	41.2 μs
Latency max	127.3 μs

The p99 latency of 41.2 μs corresponds to VM Jezgra operations (4% of traffic). The maximum latency of 127.3 μs was observed during a single garbage collection pause and is not representative of steady-state performance.

7.3 Compression Ratio

Methodology. We measured the byte size of 10,000 raw order JSON objects (as transmitted over the wire) and compared them to the corresponding 32-byte FraktalTokens.

Metric	Value
Raw input mean size	272 bytes
Raw input range	248–310 bytes
FraktalToken size	32 bytes (constant)
Compression ratio	88.37% (mean)
Compression ratio range	86.77%–89.68%

The fixed 32-byte output size ensures predictable memory allocation and network bandwidth usage regardless of input complexity.

7.4 Forgery Rejection

Methodology. We constructed 11 distinct forgery vectors (enumerated in Section 6.3) and submitted each to the verification pipeline. A successful forgery would produce a FraktalToken that passes all validation checks despite originating from an invalid or unauthorized source.

Results: 11/11 forgery attempts were rejected with HTTP 403 (Forbidden). No partial bypasses were observed — each forgery was detected at the earliest possible validation stage:

Vector	Detection Stage	Response
Random bytes	Structural validation	403
Modified anchor	Anchor verification	403
Modified payload	Determinism check	403
Modified phi_t	Temporal validation	403
Modified flags	Flag integrity	403
Revoked node	Admission check	403
Expired phi-time	Temporal validation	403
Invalid signature	Ed25519 verification	403
Replayed token	Replay detection	403
Reserved organ	Structural validation	403
Non-admitted key	Admission check	403

7.5 End-to-End Performance

Methodology. We measured round-trip time from an edge device (Android X96 Mini, connected via Honor Magic 5G mobile hotspot) to the genesis-eu verification endpoint and back. The measurement includes network transit (mobile → 5G tower → internet → IONOS Frankfurt), TLS handshake (amortized over connection), HTTP request/response serialization, and server-side processing.

Metric	Value
Ping RTT p50	146.7 ms
Ping RTT p95	203.4 ms
Server processing p50	9.96 μs
Server processing p99	48.3 μs
Network overhead (p50)	146.69 ms
Processing fraction	0.0068%

Metric	Value

Server-side processing contributes less than 0.007% of end-to-end latency. The bottleneck is network transit over the mobile connection, which is inherent to edge deployment and not addressable by protocol optimization.

7.6 Login Flow Verification

Methodology. We tested the admission-to-session bridge flow, which converts an EHO6 admission token into a genesis_auth session for authenticated API access. Four test cases were executed:

Test Case	Expected	Actual	Pass
Valid admission token	HTTP 200 + session	HTTP 200	Yes
Expired admission token	HTTP 401	HTTP 401	Yes
Forged admission token	HTTP 401	HTTP 401	Yes
Revoked node token	HTTP 401	HTTP 401	Yes

The bridge gate latency (time from token presentation to session issuance) was measured at 3,337 μs (3.34 ms), dominated by database lookup for admission status verification.

7.7 Determinism Verification

Methodology. We processed identical input sets on two independent nodes (genesis-eu and genesis-de) and compared the output FraktalTokens byte-by-byte.

Comparison	Pairs Tested	Bit-Identical	Divergent
EU $\leftrightarrow$ DE (POSAO 0.1)	12	12	0
Seed reproducibility	1,000	1,000	0
Cross-platform (Linux $\leftrightarrow$ Windows)	12	12	0

Zero divergences were observed across all test conditions, confirming the determinism guarantee (Theorem 1).

8 Comparison with Related Work

8.1 Quantitative Comparison

System	Processing Latency	Auditability	Decentralization	Compression	Token Size
EHO6	**3.52 μs**	**100%**	**Yes (multi-anchor)**	**88.37%**	**32B**
Tenderly	2–3 s	80%	No (centralized)	N/A	Variable
Forta	1–2 s	85%	Partial	N/A	Variable

System	Processing Latency	Auditability	Decentralization	Compression	Token Size
Ethereum block	12 s	0% (opaque)	Yes (PoS)	N/A	Variable

EHO6 achieves latency improvements of 5–6 orders of magnitude compared to existing systems while providing strictly superior auditability and comparable or better decentralization.

8.2 Qualitative Analysis

8.2.1 Versus Tenderly

Tenderly provides transaction simulation and debugging for EVM-compatible chains. Its verification model relies on centralized infrastructure operated by Tenderly Inc., creating a single point of failure and trust. Tenderly's latency (2–3 seconds) reflects RPC round-trips to archive nodes. EHO6 eliminates this dependency by performing verification locally with pre-computed anchor hashes.

Tenderly achieves 80% auditability because its simulation traces are comprehensive but proprietary — users must trust Tenderly's infrastructure to faithfully report simulation results. EHO6 achieves 100% auditability because every FraktalToken is independently verifiable using only the token itself and the issuing node's public key.

8.2.2 Versus Forta

Forta is a decentralized monitoring network where independent bot operators scan blockchain transactions for threats. Forta achieves partial decentralization: bot operators are independent, but alert aggregation and threat scoring rely on centralized Forta governance. EHO6's multi-anchor admission provides stronger decentralization guarantees because no single entity controls admission or verification.

Forta's 1–2 second latency reflects the time between block production and alert emission. EHO6 operates on pre-block data (raw transactions) and does not wait for block confirmation.

8.2.3 Versus Ethereum On-Chain Verification

Ethereum smart contracts provide full decentralization via proof-of-stake consensus but impose a 12-second block time floor on verification latency. On-chain verification is opaque in the sense that external observers see only the final state transition (success/revert), not the verification logic. EHO6's FraktalToken encodes the verification result in a structured, inspectable format.

Ethereum's gas costs ($\approx$ \$0.50–\$50 per verification depending on complexity and network congestion) are 3–5 orders of magnitude higher than EHO6's per-verification fee of \$0.00031.

8.2.4 Versus Zero-Knowledge Proof Systems

ZK-SNARK and ZK-STARK systems (Polygon zkEVM, zkSync, StarkNet) achieve succinctness (small proof size) and privacy (no input disclosure). However, they impose significant prover overhead: generating a ZK proof for a single transaction requires 100 ms to 10+ seconds of computation on specialized hardware.

EHO6 does not provide zero-knowledge properties — the FraktalToken's payload is deterministically derived from the input, not hidden. This is a deliberate design choice: EHO6 targets verification, not privacy. Privacy is handled at the economic layer (Monero) rather than the verification layer.

8.3 Positioning

EHO6 occupies a unique position in the design space: it provides the latency of in-process computation, the auditability of structured proofs, the decentralization of multi-party protocols, and the economic alignment of fee-based systems — all in a 32-byte fixed-size token.

The protocol is not a replacement for blockchain consensus; it is a complement. EHO6 verifications can be anchored to any settlement layer (Ethereum, Monero, Bitcoin) by including the anchor hash in on-chain transactions when permanent settlement is required.

9 Conclusion

We have presented EHO6, a deterministic semantic compression protocol for decentralized verification. The protocol's core contribution is the FraktalToken: a 32-byte structure that compresses arbitrary verification inputs (typically 272+ bytes) into a fixed-size, self-verifying token with 88.37% compression ratio.

The DNK three-layer architecture routes 69.2% of operations through a branch-free Ribosom path, achieving 3.52 μs median processing latency and 67,501 operations per second on commodity hardware. The remaining operations are handled by Regulacija (26.8%, policy enforcement) and VM Jezgra (4.0%, Turing-complete execution), with the weighted aggregate maintaining sub-4 μs median latency.

Multi-anchor admission using Ed25519 threshold signatures ($k = 2$) provides Sybil resistance without proof-of-work energy waste. The Monero economic layer aligns incentives through admission fees (0.001 XMR) and per-verification fees (0.000001 XMR) while preserving participant privacy.

All results have been validated on production hardware with deterministic reproducibility (seed=42). Forgery resistance has been demonstrated across 11 attack vectors with 100% rejection rate. Cross-node determinism has been verified with bit-identical output across independent nodes on different continents.

9.1 Future Work

Several directions for future development are under investigation:

1. **Threshold increase.** Increasing k from 2 to 3-of-5 for stronger anchor compromise resistance as the network grows.
2. **Hardware acceleration.** FPGA implementation of the Ribosom layer for sub-microsecond latency on dedicated verification appliances.
3. **Cross-chain anchoring.** Periodic Merkle root publication to Bitcoin and Ethereum for settlement finality.
4. **Mobile-native SDK.** FraktalToken generation directly on mobile devices (ARM Cortex-A) to eliminate network latency for local verification scenarios.
5. **Formal verification.** Machine-checked proofs of determinism and unforgeability using Coq or Lean 4.

10 References

1. D. J. Bernstein, N. Duif, T. Lange, P. Schwabe, and B.-Y. Yang, "High-speed high-security signatures," *Journal of Cryptographic Engineering*, vol. 2, no. 2, pp. 77–89, 2012. (Ed25519)
2. S. Josefsson and I. Liusvaara, "Edwards-Curve Digital Signature Algorithm (EdDSA)," RFC 8032, Internet Engineering Task Force, January 2017.
3. National Institute of Standards and Technology, "Secure Hash Standard (SHS)," FIPS PUB 180-4, August 2015. (SHA-256)
4. National Institute of Standards and Technology, "SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions," FIPS PUB 202, August 2015. (SHA3-256)
5. S. Noether, A. Mackenzie, and the Monero Research Lab, "Ring Confidential Transactions," *Ledger*, vol. 1, pp. 1–18, 2016.
6. N. van Saberhagen, "CryptoNote v 2.0," whitepaper, October 2013. (Monero predecessor)
7. V. Buterin, "Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform," whitepaper, 2014.
8. Forta Network, "Forta: The Decentralized Monitoring Network for Web3," whitepaper, 2021.
9. D. E. Knuth, *The Art of Computer Programming, Volume 1: Fundamental Algorithms*, 3rd ed. Addison-Wesley, 1997, pp. 79–86. (Fibonacci sequence properties)
10. R. L. Graham, D. E. Knuth, and O. Patashnik, *Concrete Mathematics*, 2nd ed. Addison-Wesley, 1994. (Number-theoretic properties of Fibonacci numbers)
11. S. Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," whitepaper, 2008.
12. M. Bellare and P. Rogaway, "Random Oracles are Practical: A Paradigm for Designing Efficient Protocols," in *Proc. ACM CCS*, 1993. (Hash function security model)
13. J. Bonneau, A. Miller, J. Clark, A. Narayanan, J. A. Kroll, and E. W. Felten, "SoK: Research Perspectives and Challenges for Bitcoin and Cryptocurrencies," in *Proc. IEEE S&P*, 2015.