ZAP: Zero-Copy Application Protocol for Contested, Edge, and Bandwidth-Limited Military Environments

Zach Kelling — Hanzo Team

Version 1.0 — February 12, 2026 February 2026

Abstract

We present ZAP (Zero-copy Application Protocol), a high-performance binary serialization and RPC framework achieving 2.2 ns parse latency with zero memory allocations, 11.5 million transactions per second end-to-end, and 91% memory reduction compared to JSON-RPC. ZAP provides two production implementations—Rust (AI agent communication, MCP gateway bridging) and Go (blockchain consensus wire protocol, VM-to-VM messaging)—sharing a common wire format with 16-byte headers and 8-byte aligned structs for direct pointer-arithmetic reads from network buffers. The protocol includes native post-quantum transport security (ML-KEM-768 + ML-DSA-65 hybrid handshake), decentralized identity via W3C DIDs derived from lattice public keys, distributed agent consensus with threshold voting, and mDNS-based ad-hoc network discovery. We analyze ZAP's suitability for contested electromagnetic environments, bandwidth-limited tactical links, GPU cluster coordination, and edge AI agent orchestration, demonstrating 96% energy reduction and 30× throughput improvement over standard protocols under identical network conditions.

Executive Summary. ZAP is the "plumbing" that lets AI agents, sensors, and computers talk to each other using far less bandwidth, memory, and power than today's standard formats. It matters most at the tactical edge: it moves the same information using roughly a tenth of the data, which both cuts the radio footprint an adversary can detect and lets coordination work over links as slow as an HF radio that ordinary protocols cannot use at all. Its security is post-quantum by design—traffic encrypted with ZAP stays protected even against a future quantum computer—and that protection is built into the protocol rather than bolted on. Concretely, it parses a message in 2.2 nanoseconds, handles 11.5 million transactions per second, and uses 91% less memory and 96% less energy than the common JSON-RPC approach, which on a battery-powered device translates directly into longer mission endurance. For program leaders, ZAP is the efficiency and security layer that makes real-time, machine-speed AI coordination feasible in contested, bandwidth-starved, and disconnected environments.

Contents

1 Introduction

Military networking at the tactical edge operates under constraints that expose fundamental weaknesses in standard data interchange protocols. JSON-RPC, gRPC/Protocol Buffers, and REST APIs assume abundant bandwidth, low latency, and the availability of garbage-collected runtimes. These assumptions fail in contested electromagnetic environments where every byte transmitted increases detection probability, every microsecond of processing delays decision cycles, and every memory allocation risks unpredictable latency spikes.

ZAP addresses these constraints through three design principles:

1. **Zero-copy reads**: Data is accessed directly from network buffers via pointer arithmetic. No deserialization pass, no object construction, no memory allocation.
2. **Minimal wire overhead**: 16-byte headers with 8-byte aligned struct fields. No field tags, no length-delimited strings for known schemas.
3. **Native PQ security**: Post-quantum key exchange and authentication built into the protocol handshake, not layered on top.

1.1 Why Existing Protocols Fail at the Edge

Protocol	Parse	Allocs	Bytes/op	Edge Failure Mode
JSON-RPC	5,579 ns	58	2,826	GC pauses, verbose encoding
Protobuf	500 ns	11–121	1,200	Varint overhead, copy-on-parse
gRPC	800 ns	30+	1,500	HTTP/2 framing, TLS overhead
Cap'n Proto	∼100 ns	0	800	No PQ crypto, complex schema
ZAP	**2.2 ns**	**0**	**256**	**None**

Table 1: Protocol comparison for tactical edge deployment.

1.2 Contributions

1. **Zero-copy wire format**: 16-byte header, 8-byte aligned structs, direct buffer reads at 2.2 ns.
2. **Dual implementation**: Rust (AI agents, MCP gateway) and Go (consensus, VMs) sharing identical wire format.
3. **Native PQ transport**: ML-KEM-768 + ML-DSA-65 hybrid handshake integrated at protocol level.
4. **Agent consensus**: Distributed voting with threshold agreement and DID-based identity.
5. **Ad-hoc discovery**: mDNS for tactical network formation without infrastructure.
6. **Environmental analysis**: 96% energy reduction enabling sustainable edge operation.

2 Wire Format Specification

2.1 Header

Every ZAP message begins with a 16-byte little-endian header:

Field	Offset	Size	Description
Magic	0	4 B	`ZAP\x00` (0x5A415000)
Version	4	2 B	Protocol version (currently 1)
Flags	6	2 B	Bitfield (see below)
Root Offset	8	4 B	Byte offset to root struct
Size	12	4 B	Total message size in bytes

Table 2: ZAP 16-byte wire header.

2.1.1 Flag Bits

Flag	Value	Meaning
`FlagNone`	0x0000	No flags set
`FlagCompressed`	0x0001	Payload is compressed
`FlagEncrypted`	0x0002	Payload is encrypted
`FlagSigned`	0x0004	Payload carries PQ signature

Table 3: ZAP header flag definitions.

2.2 Type System

Type	Wire Size	Access	Zero-Copy
Bool	1 B	Direct read	Yes
Int8–Int64	1–8 B	Direct read	Yes
Float32/64	4/8 B	Direct read	Yes
Text	8 B (off+len)	Slice into buffer	Yes
Bytes	8 B (off+len)	Slice into buffer	Yes
List(T)	8 B (off+count)	Stride iteration	Yes
Struct	4+ B	Nested offset	Yes

Table 4: ZAP type system with zero-copy access patterns.

All struct fields are 8-byte aligned, enabling direct pointer-arithmetic reads on any architecture without alignment fixups.

2.3 Zero-Copy Access Pattern

Listing 1: Zero-copy message access (Go implementation).

```
// No allocation, no copying, no deserialization
```

```
data := networkBuffer[:msgSize]
msg, _ := zap.Parse(data)  // 2.2 ns
root := msg.Root()

// Direct reads from buffer via offset calculation
nodeID := root.Uint64(0)      // 8 bytes at offset 0
timestamp := root.Int64(8)    // 8 bytes at offset 8
payload := root.Bytes(16)     // slice into buffer
```

The `Parse` function validates the header (magic, version, size bounds) and returns a handle to the root struct. No data is copied; all subsequent reads use offset arithmetic into the original buffer.

3 Schema Language

3.1 Whitespace-Significant Format

ZAP defines a clean, whitespace-significant schema language (preferred over Cap'n Proto `.capnp` format):

Listing 2: ZAP schema definition.

```
struct ConsensusVote
  validator_id  UInt64
  block_hash    Bytes
  round         UInt32
  signature     Bytes
  pq_signature  Bytes    # ML-DSA-65 (optional)

enum VoteType
  prevote
  precommit
  commit

interface ConsensusService
  vote (v ConsensusVote) -> (accepted Bool)
  get_block (hash Bytes) -> (block Bytes)
```

3.2 Compiler

The `zapc` compiler generates code for multiple target languages:

- **Rust**: Primary target, zero-copy readers and builders.
- **Go**: Used in Hanzo consensus and VM communication.
- **Python, TypeScript**: SDK generation for agent integration.
- **C, C++, Java**: Available for embedded and enterprise integration.

The compiler also supports legacy `.capnp` files with automatic migration via `capnp_to_zap()`.

4 Post-Quantum Transport Security

4.1 Cryptographic Primitives

Algorithm	Purpose	Key/CT Size
ML-KEM-768 (FIPS 203)	Key encapsulation	PK: 1,184 B, CT: 1,088 B
ML-DSA-65 (FIPS 204)	Digital signatures	PK: 1,952 B, Sig: 3,309 B
X25519	Classical ECDH	PK: 32 B
HKDF-SHA256	Key derivation	—
AES-256-GCM	Session encryption	Key: 32 B

Table 5: ZAP cryptographic primitives.

4.2 Hybrid Handshake Protocol

Algorithm 1 ZAP Post-Quantum Hybrid Handshake

Initiator generates ephemeral keys:
  $(\text{ek}^X, \text{epk}^X) \leftarrow$ X25519.Generate()
  $(\text{ek}^M, \text{epk}^M) \leftarrow$ ML-KEM-768.Generate()
Send: $\text{epk}^X \| \text{epk}^M \| \text{pk}^{\text{DSA}} \| \sigma$
  $\sigma =$ ML-DSA-65.Sign(sk^{DSA}, transcript)

Responder verifies σ, then:
  $\text{ss}^X \leftarrow$ X25519.DH($\text{sk}^X, \text{epk}^X$)
  $(\text{ct}^M, \text{ss}^M) \leftarrow$ ML-KEM-768.Encaps(epk^M)
  $K \leftarrow$ HKDF-SHA256($\text{ss}^X \| \text{ss}^M$)
Send: $\text{epk}_B^X \| \text{ct}^M \| \sigma_B$

Both: Derive session keys from K
Both: Destroy ephemeral private keys (forward secrecy)
Session: AES-256-GCM with counter nonces

4.3 Crypto Backend Chain

The Rust implementation supports two backends with automatic fallback:

1. **Post-quantum crypto FFI library** (preferred): FFI binding to the Hanzo cryptographic library (CIRCL-based, CGO-optimized). Shared with the consensus layer.

2. **pqcrypto** (fallback): Pure Rust PQ implementations for environments without the native crypto library.

3. **x25519-dalek**: Classical ECDH (always present for hybrid mode).

4.4 Handshake Overhead Analysis

Component	Size	Classical	PQ Hybrid
Ephemeral X25519 PK	32 B	✓	✓
Ephemeral ML-KEM PK	1,184 B	—	✓
ML-KEM ciphertext	1,088 B	—	✓
ML-DSA-65 PK	1,952 B	—	✓
ML-DSA-65 signature	3,309 B	—	✓
Per-direction		**32 B**	**~7.5 KB**
Amortized/message		**0 B**	**0 B**

Table 6: Handshake overhead: one-time cost, zero per-message overhead.

The 7.5 KB handshake is a *one-time* cost per connection. All subsequent messages use AES-256-GCM with 28-byte overhead (12 B nonce + 16 B auth tag), identical to classical TLS.

5 Performance Analysis

5.1 Microbenchmarks

5.1.1 Go Implementation (Consensus Wire Protocol)

Operation	Time	Ops/sec	Allocs
Parse	2.2 ns	446M	0
Build	38.9 ns	25.7M	0
Consensus round	4.6 ns	218M	0

Table 7: ZAP Go implementation benchmarks (10-core, Go 1.26.1).

5.1.2 Rust Implementation (AI Agent Communication)

Operation	Time	Ops/sec	Allocs
Serialize tool call	322 ns	3.1M	2
Deserialize response	21 ns	47.6M	0
MCP gateway route	1.2 μs	833K	4
Agent consensus vote	890 ns	1.12M	2

Table 8: ZAP Rust implementation benchmarks.

Note: The Go and Rust implementations benchmark different operations. The Go "Parse" (2.2 ns) reads a pre-built message from buffer. The Rust "Serialize" (322 ns) constructs a new message. The Rust "Deserialize" (21 ns) performs schema-validated parsing, more expensive than raw buffer validation.

5.2 End-to-End Throughput

Scenario	ZAP	JSON-RPC
Single-thread serial	3.1M ops/s	179K ops/s
20-server MCP gateway	4.2M calls/s	140K calls/s
Consensus (5 validators)	11.5M TPS	246K TPS

Table 9: End-to-end throughput comparison.

5.3 Memory Efficiency

Metric (100K ops)	JSON-RPC	ZAP
Total allocated	304.01 MB	26.70 MB
Malloc count	6,001,513	200,003
Bytes/op	3,187	280
Mallocs/op	60	2
GC pauses (50K ops)	41	3
Memory saved	**277.3 MB (91.2%)**	
Allocs saved	**5,801,510 (96.7%)**	

Table 10: Memory profile: ZAP vs. JSON-RPC over 100K operations.

5.4 Environmental Impact

At scale (1 billion tool calls per day):

Metric	JSON-RPC	ZAP	Savings
Memory throughput	2.9 TB/day	0.3 TB/day	2.6 TB
Energy consumption	1.8 kWh/day	0.1 kWh/day	96%
CO_2 emissions	∼0.7 kg/day	∼0.03 kg/day	∼245 kg/year

Table 11: Environmental impact at 1B calls/day.

For battery-powered tactical edge devices, the 96% energy reduction translates directly to extended operational endurance.

6 Transport Layer

6.1 Supported Transports

Scheme	Transport	Status	Use Case
`zap://`	TCP	Complete	Default RPC
`tcp://`	TCP explicit	Complete	Direct TCP
`unix://`	Unix socket	Complete	Local IPC
`ws://`	WebSocket	Complete	Browser/cloud
`wss://`	WebSocket+TLS	Complete	Secure browser
`stdio://`	Stdio	Complete	MCP subprocess
`http://`	HTTP/SSE	Complete	Remote MCP
`udp://`	UDP	Complete	Low-latency

Table 12: ZAP transport schemes.

Frame format: 4-byte big-endian length prefix + payload (max 16 MB).

6.2 mDNS Discovery for Ad-Hoc Networks

ZAP nodes advertise services via mDNS with domain-specific service types:

Service Type	Purpose
`_hzd._tcp`	Blockchain consensus nodes
`_hz-gpu._tcp`	GPU compute nodes
`_hz-vm._tcp`	Virtual machine endpoints
`_zap-agent._tcp`	AI agent endpoints

Table 13: mDNS service types for ad-hoc discovery.

This enables zero-configuration tactical network formation: devices joining a mesh automatically discover available services without DNS infrastructure or pre-configured endpoints.

7 Decentralized Identity

7.1 W3C DID Support

ZAP implements W3C Decentralized Identifiers with three methods:

- **did:key**: Self-certifying from ML-DSA-65 public keys. No external registry required.
- **did:hanzo**: Anchored to the Hanzo chain. Verifiable via consensus.
- **did:web**: DNS-based for interoperability with existing PKI.

Listing 3: DID generation from ML-DSA public key.

```
// Post-quantum self-certifying identity
let identity = Identity::generate_pq()?;
let did = identity.to_did();
// did:key:z6Mk... (multibase-encoded ML-DSA-65 pubkey)
```

7.2 Stake-Weighted Trust

For consensus operations, DIDs can be associated with stake weights:

- Validators register stake via blockchain transaction.
- Agent consensus votes are weighted by stake.
- Sybil resistance: creating identity is free, but gaining voting weight requires stake.

8 Agent Consensus Protocol

8.1 Distributed Agent Voting

When multiple AI agents must agree on a decision (threat classification, course of action), ZAP provides distributed consensus:

Algorithm 2 ZAP Agent Consensus

Initiator broadcasts query Q to n agents
for each agent A_i **do**
    A_i computes response R_i independently
    A_i signs: $\sigma_i = \text{ML-DSA-65.Sign}(\text{sk}_i, Q \| R_i)$
    A_i returns $(R_i, \sigma_i, \text{DID}_i)$
end for
Collect responses until threshold t reached (default $t = \lceil 2n/3 \rceil$)
Verify all signatures
Determine consensus: majority response among verified votes
if consensus reached ($\geq t$ agree) **then**
    return (ConsensusResult, aggregated signatures)
else
    return (NoConsensus, individual responses)
end if

8.2 Defense Application

Agent consensus is critical for high-consequence decisions:

- **Threat classification**: Multiple AI models independently assess a sensor contact; consensus prevents single-model hallucination from triggering false alarms.
- **Targeting**: Distributed analysis with threshold agreement before engagement recommendation.
- **Intelligence fusion**: Multiple agents process different data sources; consensus identifies corroborated intelligence.

9 MCP Gateway

9.1 Multi-Server Aggregation

The ZAP MCP gateway bridges multiple Model Context Protocol servers into a unified tool surface:

- **Tool namespacing**: Each server's tools prefixed (e.g., `filesystem:read_file`).
- **Health checking**: Automatic reconnection on server failure.
- **Resource aggregation**: Resources and prompts from all servers accessible through single endpoint.
- **Protocol bridging**: Stdio, HTTP/SSE, WebSocket, and native ZAP servers unified.

9.2 Performance Advantage

Metric	MCP JSON-RPC	ZAP Gateway
Tool calls/sec (20 servers)	140,000	4,200,000
Serialization overhead	5,579 ns	322 ns
Memory per call	2,826 B	256 B

Table 14: MCP gateway performance: native vs. ZAP-bridged.

10 GPU Cluster Coordination

10.1 Zero-Copy Ciphertext Handling

ZAP's zero-copy design enables direct GPU ciphertext operations:

Listing 4: GPU FHE operation via ZAP.

```
gpuNode := zap.NewNode(zap.NodeConfig{
    NodeID:      "gpu-node-1",
    ServiceType: "_hz-gpu._tcp",
    Port:        7000,
})

gpuNode.Handle(MsgTypeFHEOp,
  func(ctx context.Context, from string,
       msg *zap.Message) (*zap.Message, error) {
    root := msg.Root()
    opType := root.Uint32(0)     // FHE operation
    ct := root.Bytes(4)          // Zero-copy ciphertext

    // GPU kernel dispatch - no memcpy needed
    result := gpuFHE(opType, ct)
    return buildResult(result), nil
})
```

Ciphertext data flows from network buffer to GPU kernel without intermediate copies, critical for FHE workloads where ciphertexts are 100×–1000× larger than plaintext.

11 Contested Environment Analysis

11.1 Low-Probability-of-Intercept (LPI)

Protocol	Bytes/call	RF Exposure	LPI Factor
JSON-RPC	2,826	100% (baseline)	1.0×
gRPC	1,500	53%	1.9×
Protobuf	1,200	42%	2.4×
ZAP	**256**	**9.1%**	**11×**

Table 15: RF exposure reduction per operation.

ZAP reduces RF exposure by 11× compared to JSON-RPC, directly improving low-probability-of-intercept and low-probability-of-detection (LPI/LPD) performance for tactical radio links.

11.2 Bandwidth-Limited Links

For a 9.6 kbps HF radio link:

Protocol	Calls/sec at 9.6 kbps	Latency/call
JSON-RPC	0.42	2.35 s
Protobuf	1.00	1.00 s
ZAP	**4.69**	**0.21 s**

Table 16: Throughput on 9.6 kbps HF radio link.

ZAP enables near-real-time AI agent coordination over HF radio links that would be unusable with JSON-RPC.

11.3 Jamming Resilience

- **Minimal transmission time**: 91% less data reduces jamming window.
- **Burst-compatible**: Small message size enables burst transmission between jamming pulses.
- **mDNS re-discovery**: Network reforms around jammed nodes via multicast DNS.
- **Multi-path**: RNS mesh provides alternative routes when primary links are degraded.

11.4 Electronic Attack Resistance

- **PQ encryption**: Intercepted traffic immune to quantum cryptanalysis.
- **Forward secrecy**: Each session uses ephemeral keys; compromising one session reveals nothing about others.
- **Zero metadata leakage**: Binary format reveals no field names, no schema structure, no type information to passive observers.
- **Traffic analysis resistance**: Fixed-size messages possible via padding flag.

12 Comparison with Military Standards

Feature	**Link 16 (TADIL-J)**	**VMF**	**ZAP**
Encoding	Fixed-word binary	Variable-length binary	Zero-copy binary
Bandwidth	238 kbps	Variable	Wire rate
Latency	12.8 ms slot	Variable	2.2 ns parse
Crypto	Type 1	HAIPE	PQ hybrid (FIPS)
AI Support	None	None	Native (agents, MCP)
Discovery	JTIDS	IP-based	mDNS ad-hoc
PQ-Safe	No	No	Yes (FIPS 203/204)

Table 17: ZAP vs. existing military data link protocols.

ZAP is not a replacement for Link 16 or VMF—it operates at a different layer (application vs. tactical data link). However, ZAP can serve as the application protocol carried *over* these links, providing AI agent coordination, consensus messaging, and PQ security that existing message standards lack.

13 Corona Consensus Integration

13.1 Post-Quantum Threshold Signing

ZAP integrates with the Corona protocol for distributed PQ threshold signatures:

- **Basis**: Practical two-round threshold signature from LWE (IACR ePrint 2024/1113).
- **Parameters**: $M = 8, N = 7, \bar{D} = 48, \kappa = 23, Q =$ `0x1000000004A01`.
- **Security**: NIST Level 3 (192-bit equivalent) under Ring-LWE hardness.
- **Performance**: $\sim$89 ms for threshold signature (within 3-second quantum bundle window).

ZAP consensus messages carry Corona shares as native struct fields, enabling post-quantum finality for distributed agent decisions.

14 Implementation Details

14.1 Rust Crate

Module	Size	Purpose
`client.rs`	28.5 KB	ZAP RPC client
`server.rs`	32.2 KB	ZAP RPC server
`gateway.rs`	45.2 KB	MCP gateway aggregator
`transport.rs`	34.6 KB	TCP, WS, stdio, UDP
`crypto.rs`	28.3 KB	ML-KEM, ML-DSA, X25519
`consensus.rs`	35.5 KB	Corona threshold signing
`identity.rs`	26.5 KB	W3C DIDs
`schema.rs`	79.9 KB	Schema compiler
`agent_consensus.rs`	14.2 KB	Agent voting protocol
Total	**∼9,915 lines**	

Table 18: ZAP Rust implementation structure.

14.2 Go Implementation

Minimal-dependency implementation for consensus and VM communication:

- `zap.go`: Zero-copy builder and parser (8.9 KB).
- `node.go`: mDNS discovery and async RPC (21.2 KB).
- `mcp/`: MCP bridge for 20+ tool servers.
- Single external dependency: `github.com/hanzoai/mdns`.

15 Strategic Context

ZAP is the transport layer of an integrated, sovereign AI stack built by Hanzo, an American applied-AI lab founded in 2014 (Techstars '17) whose founder works at the frontier of both AI/ML and cryptography. American frontier AI has consolidated into a closed monoculture—effectively two labs reachable only as metered cloud services—while the only openly available frontier model weights today are largely foreign; Hanzo is the American open-source frontier alternative, owning both the model (the Zen family, 0.8B–1T+ parameters, with owned weights) and the cryptography (the production post-quantum stack ZAP is part of). That ownership is what lets the whole stack run sovereign and disconnected, where the data lives. ZAP also supplies a measurable piece of the economics: its 91% memory and 96% energy reductions and 30× throughput, combined with owned models (no per-token API markup) and self-hosting (no hyperscaler margin), are the mechanism behind roughly an order-of-magnitude lower infrastructure cost in aggregate. The efficiency is not academic—the same agent-driven stack this protocol carries was used to deliver a FINRA/SEC-approved, SOC2-compliant securities platform to production—a full ATS/BD/TA stack consolidated from 200+ microservices into 3 compiled binaries—in record time with a small team, showing the approach holds up under the demands of a regulated, audited domain.

16 Conclusion

ZAP provides the fastest possible binary protocol for military edge environments, achieving 2.2 ns parse latency, zero memory allocations, and 11× RF exposure reduction compared to standard protocols. Its native post-quantum transport security (FIPS 203/FIPS 204), ad-hoc mDNS discovery, distributed agent consensus, and dual Rust/Go implementations make it uniquely suited for contested, bandwidth-limited, and disconnected tactical operations.

The protocol's 96% energy reduction extends battery endurance on tactical devices, while its zero-copy design eliminates the GC pauses and memory pressure that plague JSON-RPC and Protobuf in resource-constrained environments. ZAP enables real-time AI agent coordination over links as slow as 9.6 kbps HF radio—a capability no existing protocol provides.

References

[1] NIST, "ML-KEM Standard," FIPS 203, 2024.

[2] NIST, "ML-DSA Standard," FIPS 204, 2024.

[3] "Practical Two-Round Threshold Signature from LWE," IACR ePrint 2024/1113, 2024.

[4] K. Varda, "Cap'n Proto: Insanely Fast Data Interchange Format," 2013.

[5] Google, "FlatBuffers: Memory Efficient Serialization Library," 2014.

[6] Google, "Protocol Buffers: Language-Neutral Data Serialization," 2008.

[7] Anthropic, "Model Context Protocol Specification," 2024.

[8] W3C, "Decentralized Identifiers (DIDs) v1.0," 2022.

[9] S. Cheshire, M. Krochmal, "Multicast DNS," RFC 6762, 2013.

[10] DoD, "Tactical Digital Information Link (TADIL) J Standard," MIL-STD-6016, 2018.

[11] DoD, "Variable Message Format (VMF)," MIL-STD-6017, 2019.

[12] NSA, "CNSA Suite 2.0," 2022.

[13] R. Barnes *et al.*, "Hybrid Public Key Encryption," RFC 9180, 2022.

[14] Cloudflare, "CIRCL: Interoperable Reusable Cryptographic Library," 2024.

[15] Hanzo Team, "Quasar Consensus," Hanzo, 2025.

[16] Hanzo Team, "ZAP: Consensus Wire Protocol," Hanzo, 2026.

Reproducibility

Source code. `github.com/hanzoai/zap` (commit `TBD`).
Build. cargo build –release (Rust impl); go build ./... (Go impl).
Hardware tested. Apple M-series, x86_64 Linux + NVIDIA A100, ARM Cortex-A (edge).
Standards referenced. Cap'n Proto wire spec, ML-KEM-768, ML-DSA-65 (FIPS 203/204), W3C DID.
Contact. research@hanzo.ai.