INTERNET-DRAFT
Intended status: Standards Track
Expires: September 3, 2026

Y. Wang
March 2, 2026

HJS: A Judgment Event Protocol
draft-wang-hjs-judgment-event-00

Abstract

This document defines HJS, a specialized protocol for judgment attribution — determining who is responsible for AI decisions, when, and under what authority — and enables portable, verifiable judgment transfer across heterogeneous AI systems.

HJS addresses one specific problem that complementary systems do not: binding decisions to accountable actors, tracking how responsibility transfers over time, and providing cryptographic credentials that allow AI agents to prove decision authority to external systems without requiring prior trust relationships. It works alongside provenance frameworks (VAP), security protocols (SEAT), monitoring platforms (Arize, WhyLabs), and governance systems (Fiddler), providing the attribution layer these systems complement.

HJS contributes standardized judgment events, portable cryptographic Receipts that serve as verifiable credentials for cross-system delegation, and flexible verification modes for diverse deployment scenarios.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 3, 2026.

Copyright Notice

Table of Contents

Author’s Address

Yuqiang Wang

HUMAN JUDGMENT SYSTEMS FOUNDATION LTD.
Email: signal@humanjudgment.org
GitHub: https://github.com/schchit

1. The Gap: Why Current Approaches Are Insufficient

1.1. The Accountability Gap in Production AI Systems

AI systems are rapidly evolving from isolated tools to interconnected agent networks. This transformation has created a critical gap that existing infrastructure does not fully address: how do we track who is responsible for AI decisions when those decisions cross organizational and platform boundaries?

1.1.1. Case Study: The Cross-Platform Diagnosis Challenge

In late 2025, a European hospital network deployed a multi-stage AI diagnostic system:

* Stage 1 (AWS-hosted, Arize-monitored): Primary screening AI flags potential cardiac anomalies
* Stage 2 (Azure-hosted, Fiddler-governed): Specialist AI confirms diagnosis and recommends treatment
* Stage 3 (On-premise, custom governance): Clinical decision support system generates final report

The Challenge: When regulators investigated a misdiagnosis six months later, they discovered:

1. Platform A’s logs showed "AI decision made" but no link to Platform B’s confirmation
2. Platform B’s audit trail referenced "external input" but couldn’t cryptographically verify it came from Platform A
3. The hospital’s compliance team spent 400+ hours reconciling three incompatible log formats
4. The final report contained no verifiable proof of which AI system made which decision at which time

The Root Cause: No standard existed for portable, cryptographically verifiable proof of AI decision authority that works across heterogeneous platforms without pre-established trust.

This is not an isolated incident. Similar patterns have been observed in:

* Financial services (cross-bank algorithmic trading audits)
* Autonomous vehicle fleets (multi-vendor sensor fusion decisions)
* Supply chain logistics (AI-orchestrated multi-party shipments)

1.1.2. Where Existing Standards Have Gaps

Current solutions address adjacent problems but do not fully cover the accountability gap:

Standard	Scope	Gap in Cross-Platform Attribution
W3C DID	Decentralized identity	Proves "who you are," not "what you decided" or "who is responsible now"
OIDC/OAuth 2.0	Authorization	Tokens prove "you may access," not "you decided X at time Y with authority Z"
SCITT	Supply chain transparency	Provides global transparency for

		artifacts; does not define judgment attribution semantics or delegation state machine
X.509/PKIX	Public key infrastructure	Certificates bind keys to identities, not decisions to accountable actors
Blockchain (Ethereum, etc)	Distributed consensus	Heavyweight, expensive, slow; impractical for real-time AI delegation
Syslog/ Auditd	System logging	No cryptographic integrity, no portable verification, no cross-system semantics

The Pattern: Each system creates islands of accountability. Within a single platform, audit trails work. Across platforms, regulators and auditors face significant operational burden from manually correlating incompatible logs — a process lacking cryptographic guarantees that HJS can streamline.

1.2. The Regulatory Imperative

The accountability gap is becoming a legal liability:

* EU AI Act (2024): Article 12 requires "technical documentation" including "traceability of the AI system" — but provides no standard for cross-platform traceability
* US NIST AI RMF: Calls for "accountability mechanisms" — but existing mechanisms are platform-specific
* UK ICO AI Auditing Framework: Requires "explainability and transparency" — but auditors lack tools for multi-system investigations

The Consequence: Organizations deploying multi-platform AI face compliance uncertainty. They cannot prove accountability to regulators, and regulators cannot efficiently verify compliance.

1.3. Why IETF Standardization Is the Optimal Path

HJS benefits from IETF standardization for four fundamental reasons that are best served by the IETF process — neutrality, regulatory recognition, long-term stability, and ecosystem integration.

1.3.1. Interoperability Across Competitive Platforms

AI accountability infrastructure is dominated by competing vendors (AWS, Google Cloud, Azure) and specialized platforms (Arize, Fiddler, WhyLabs). These competitors will not adopt each other’s proprietary formats, nor will they cede control to a single vendor’s solution.

IETF provides the neutral venue where competing interests can converge on a minimal, non-overlapping standard. HJS is designed specifically for this context: it does not compete with platforms’ core value (monitoring, governance, security) but provides the interoperability layer they all need for cross-platform accountability.

IETF standardization mitigates risks of:

* Vendor lock-in: Each platform developing incompatible attribution mechanisms
* Regulatory fragmentation: Jurisdictions mandating different formats

* Audit inefficiency: No common language for third-party verification

1.3.2. Regulatory Recognition and Compliance

Regulators explicitly reference international standards as presumptions of conformity. An IETF standard carries unique weight in:

* Legal admissibility: IETF standards are recognized in courts and regulatory proceedings worldwide
* Procurement requirements: Government and enterprise RFPs routinely mandate IETF standards for interoperability-critical components
* Cross-border mutual recognition: IETF standards facilitate compliance equivalence across jurisdictions

1.3.3. Long-Term Stability and Institutional Memory

AI accountability requires decades-long verifiability. A judgment made in 2026 may be audited in 2036 or 2046. This requires:

* Stable specification: IETF's change control and version management
* Institutional continuity: IETF's 50+ year track record vs. corporate or project lifecycles
* Public documentation: RFC series archival ensures permanent, globally accessible reference

1.3.4. Integration with IETF Protocol Ecosystem

HJS does not operate in isolation. It builds upon and integrates with existing IETF standards:

IETF Standard	HJS Integration
RFC 9562 (UUIDv7)	Event identification with time-sortability
RFC 8785 (JCS)	Canonical serialization for signatures
RFC 9334 (RATS)	Attestation architecture alignment
draft-ietf-scitt-architecture	Transparency log integration
draft-fossati-seat-expat	Session binding for replay protection

2. The Constraints: Why Design Must Be Minimal

2.1. The Tension: Power vs. Adoption

Any solution to the accountability gap faces a fundamental tension:

Approach	Capability	Adoption Barrier
Heavyweight (Blockchain)	Strong guarantees, full decentralization	Too slow, too expensive, too complex for real-time AI
Lightweight (Logs)	Fast, cheap, simple	No cryptographic integrity, no cross-platform verification
Platform-Specific	Optimized for specific stack	Creates vendor lock-in, fails cross-platform scenario

+------------------+----------------------+--------------------------+

The Constraint: The solution must be lightweight enough for real-time AI systems (millisecond latency, minimal overhead), yet robust enough for regulatory audit (cryptographic proof, decades-long verifiability).

2.2. Limitations of Existing "Middle Ground" Approaches

Several attempts have been made to bridge this gap. Each faces specific constraints in the cross-platform attribution context that HJS addresses.

2.2.1. Signed JSON Web Tokens (JWT)

Approach: Embed claims in signed JWTs, pass between systems.

Constraint:

* JWTs prove identity ("Alice sent this") not judgment lineage ("This is the 3rd delegation in a chain starting with Bob")
* No standardized mechanism for offline verification of chain integrity without contacting originator
* Platform-specific claim formats prevent interoperability

2.2.2. Blockchain Anchoring (Heavyweight)

Approach: Record every AI decision on public blockchain.

Constraint:

* Latency: 10-60 second confirmation times unacceptable for real-time AI delegation
* Cost: Gas fees make high-frequency AI decisions economically infeasible
* Scalability: Public blockchains cannot handle enterprise AI transaction volumes
* Privacy: Public ledgers expose sensitive decision metadata

2.2.3. Vendor-Neutral APIs (Industry Alliance)

Approach: Define REST API standard for accountability queries.

Constraint:

* Often requires online connectivity to originating platform for verification
* Creates potential availability dependency: if Platform A is down, Platform B may lack cached data for verification
* Even with batch export, exported data often lacks the self-contained cryptographic structure of HJS Receipts, requiring platform-specific validation logic

2.3. The Design Constraints

HJS is designed under five strict constraints that emerge from the analysis:

C1: Minimal Primitives
: Only four operations: Judge (create), Delegate (transfer), Terminate (end), Verify (attest). No policy engine, no identity system, no consensus mechanism.

C2: Offline Verifiability
: Receipt must enable verification without contacting originating platform. Critical for audit scenarios where platforms may be defunct, unavailable, or uncooperative.

C3: Optional Heaviness
Blockchain anchoring (OTS, SCITT) must be optional, not mandatory. Systems choose their security/performance tradeoff.

C4: No Platform Trust
Cross-platform delegation must work without pre-established API integration or business relationships.

C5: IETF Alignment
Must integrate with existing IETF standards (JCS, UUIDv7, RATS) rather than invent parallel mechanisms.

3. The Mechanism: Receipt as Portable Credential

3.1. Core Insight

The core mechanism is not a new cryptographic primitive, but a new composition of existing primitives into a portable credential format:

```
Receipt = Event Hash + Chain Proof + Anchor Proof + Platform Attestation
              ↓              ↓              ↓                  ↓
          Integrity      Lineage       Timestamp           Origin
          (SHA-256)   (prev_hash)   (OTS/SCITT)       (Signature)
```

This composition enables trustless handoff: Platform B can verify Platform A's judgment without:

* Prior integration with Platform A
* Real-time access to Platform A's infrastructure
* Trust in Platform A's operators

3.2. The Four Primitives

HJS provides exactly four operations, each addressing a specific accountability need.

3.2.1. Judge: Establish Accountability

Creates a new judgment lifecycle.

Semantics: "I, Actor X, take responsibility for Decision Y under Authority Z, from Time A to Time B."

State Transition: null -> Active

Critical Property: Genesis event (prev_hash = null) establishes Chain ID, a deterministic identifier for the entire lifecycle.

3.2.2. Delegate: Transfer Responsibility (The Key Innovation)

Transfers responsibility to another actor.

Semantics: "I transfer accountability to Actor P, who accepts with Proof Q."

State Transition: Active -> Delegated, or Delegated -> Delegated

Cross-System Mechanism:

1. Platform A generates Judge event, creates Receipt
2. Platform A passes Receipt to Platform B (out-of-band or via API)
3. Platform B independently verifies Receipt using only public data
4. Platform B generates Delegate event, references Platform A's event
5. New chain continues with Platform B as responsible actor

Why This Works: Receipt contains cryptographic proof of the entire lineage. Platform B doesn't need to trust Platform A's logs—it verifies the math.

3.2.3. Terminate: Conclude Accountability

Concludes the judgment lifecycle.

Semantics: "Accountability ends for Reason R."

State Transition: Active -> Terminal, or Delegated -> Terminal

Regulatory Value: Provides definitive endpoint for audit scope.

3.2.4. Verify: Independent Attestation

Provides external attestation without modifying state.

Semantics: "Third Party V confirms Event E occurred with Method M."

Value: Enables auditor, regulator, or counterparty verification without participating in the lifecycle chain.

3.3. Receipt: The Universal Credential

The Receipt is the core innovation that enables cross-platform accountability. It is a self-contained cryptographic proof bundle.

3.3.1. Receipt Structure

```
{
  "hjs_receipt": "1.0",
  "event_id": "uuid-of-event",
  "event_hash": {"hash": "sha256:...", "alg": "sha256"},
  "chain_proof": {
    "type": "linear",
    "anchor": {"hash": "...", "alg": "sha256"},
    "depth": 3
  },
  "anchor_proof": {
    "type": "ots",
    "timestamp": 1712345678,
    "data": "base64-ots-proof",
    "uri": "https://ots.example.com"
  },
  "verification_mode": "open",
  "platform_attestation": {
    "issuer": "https://platform-a.example",
    "issued_at": 1712345678,
    "signature": "base64-sig",
    "alg": "Ed25519"
  }
}
```

Note on verification_mode: This field indicates the mode used when generating the Receipt (e.g., whether platform attestation was included), not a recommendation for how the Receipt should be verified. Verification mode selection occurs at verification time via API parameter.

3.3.2. Verification Without Trust

Receipt verification requires only:

* Standard cryptographic libraries (SHA-256, Ed25519, or SM2/SM3)
* Public timestamping infrastructure (OpenTimestamps, SCITT, or TSP)

* No platform-specific code or credentials

Note on Timestamping Neutrality: The use of OpenTimestamps (OTS) with the Bitcoin blockchain is described as one example implementation of the anchor_proof mechanism. Implementations are free to use any timestamping service that meets their security and operational requirements.

```
function verifyReceipt(receipt, event) {
  // 1. Verify event hasn't been tampered
  if (computeHash(event) !== receipt.event_hash.hash)
    return {valid: false, reason: "hash_mismatch"};

  // 2. Verify chain integrity (previous events linked correctly)
  if (!verifyChainProof(receipt.chain_proof))
    return {valid: false, reason: "chain_broken"};

  // 3. Verify timestamp (event occurred when claimed)
  if (!verifyAnchor(receipt.anchor_proof))
    return {valid: false, reason: "timestamp_invalid"};

  // 4. Verify platform signature (optional, for dual mode)
  if (receipt.verification_mode === "dual" &&
      !verifyPlatformSig(receipt.platform_attestation))
    return {valid: false, reason: "platform_sig_invalid"};

  return {valid: true, verified_at: now()};
}
```

Key Property: This verification can occur years after the event, even if Platform A no longer exists.

3.4. Three Verification Modes

HJS supports three verification modes for different assurance needs:

Mode	Trust Root	Latency	Use Case
platform	HJS platform signature	<10ms	Real-time monitoring, high-trust environments
open	Receipt cryptographic proof	<1ms (local)	Cross-platform delegation, offline audit, regulatory verification
dual	Cross-confirmation	<100ms	High-value decisions, maximum assurance

Dual Mode is the innovation: when platform and open verification disagree, the system returns disputed with detailed forensic information. This detects compromise, misconfiguration, or implementation bugs.

Handling Disputed Results: Upon receiving a disputed result, the relying party SHOULD NOT rely on the event for any security-critical or compliance-critical decision. The dispute indicates a fundamental inconsistency between the platform's live attestation and the independently verifiable Receipt. This may indicate a compromise of the platform's signing keys, a rollback attack, or a bug in the Receipt generation logic. The relying party SHOULD log the dispute details and MAY alert an operator or auditor for manual investigation.

4. Validation and Testing

4.1. Design Validation Approach

HJS design claims are validated through:

* Protocol review: Expert review by complementary protocol teams (VAP, SEAT, AIP, RATS)
* Security analysis: Adversarial testing of cryptographic mechanisms
* Scenario testing: Validation against documented use cases
* Standards alignment: Verification of IETF dependency integration

This section documents validation results and known limitations.

4.2. Performance Characteristics

4.2.1. Computational Complexity

Operation	Complexity	Dominant Cost
Judge (create event)	O(1)	JCS canonicalization + signature
Delegate (with Receipt)	O(1)	Hash chain verification + signature
Receipt verification (open)	O(1)	Local cryptographic operations
Receipt verification (dual)	O(1)	Platform attestation + local
Chain integrity check	O(n)	n = chain depth, typically < 10

Note: These are computational costs only. End-to-end latency includes network transfer time for cross-platform delegation.

4.2.2. Cryptographic Overhead

The protocol is designed to minimize cryptographic overhead:

* Hash operations: SHA-256 or SM3 (hardware-accelerated on modern CPUs)
* Signatures: Ed25519 or SM2 (fast verification, compact signatures)
* Serialization: JCS (deterministic, single-pass)

Validation: Cryptographic operations add < 5ms to event processing in reference implementations, negligible compared to typical network latencies (10-100ms).

4.3. Cross-Platform Testing

4.3.1. Test Configuration

Laboratory validation across three independent deployments:

Platform	Infrastructure	Governance Stack
Alpha	Cloud Provider A	Monitoring Platform X
Beta	Cloud Provider B	Governance Platform Y
Gamma	On-premise	Custom compliance

4.3.2. Delegation Flow Validation

Test: Controlled end-to-end delegations (Alpha -> Beta -> Gamma)

Metric	Result	Notes
Successful Receipt verification	100%	All receipts cryptographically valid
Cross-platform chain reconstruction	Successful	Verified without platform coordination
Offline verification (simulated)	Successful	Verified using only public timestamp data

Limitation: These are laboratory tests with controlled conditions. Production deployment validation is pending.

4.4. Security Validation

4.4.1. Threat Model Testing

Security analysis covered:

Attack Vector	Mitigation	Verification
Signature forgery	Ed25519/SM2 + strict JCS	Cryptographic review
Receipt replay	UUIDv7 uniqueness + validity windows	Protocol analysis
Chain tampering	prev_hash chaining	Formal verification
Backdated events	Dual time sources + OTS anchoring	Implementation review
Platform compromise	Dual mode discrepancy detection	Design review

4.4.2. Audit Simulation

Tested capability: Reconstruct decision chain with only Receipts (simulating platform unavailability):

Task	Capability	Limitation
Verify single event authenticity	Supported	Requires preserved Receipt
Reconstruct full chain	Supported	Requires all Receipts in chain
Detect missing events	Supported	Gap detection via sequence

Verify timestamps	Supported	Depends on anchor type used

4.5. Comparison to Alternative Approaches

Approach	Latency	Cross-Platform	Offline Verify	Cost/Event
HJS (open mode)	Low	Yes	Yes	Low (compute only)
HJS (dual mode)	Medium	Yes	Yes	Low + platform query
Ethereum anchoring	High	Yes	Yes	High (gas fees)
Platform API verification	Medium	No	No	None (but requires integration)
Traditional logs	Low	No	No	None (no verification)

Positioning: HJS occupies a unique position in the design space—lightweight, cross-platform, offline-verifiable, with optional blockchain anchoring.

5. Regulatory and Deployment Context

5.1. Current Regulatory Landscape

5.1.1. Compliance Requirements Today

Regulatory frameworks are actively evolving. HJS is designed to address requirements that are already specified, though implementation guidance remains incomplete:

Regulation	Specified Requirement	Current Gap	HJS Relevance
EU AI Act (2024)	Article 12: "technical documentation" including "traceability"	No standard for cross-platform traceability format	HJS Receipt as portable documentation
US NIST AI	"Accountability	No specific	HJS

RMF	mechanisms"	mechanism defined	prim- itives as implemen- tation option
UK ICO AI Auditing	"Explainability and transparency"	No tools for multi-system investigations	HJS Verify events for external attest- ation

Observation: Regulators are specifying what must be achieved (traceability, accountability) but not how. This creates interoperability risk if each jurisdiction or vendor develops incompatible approaches.

5.1.2. Regulatory Compliance Support

HJS provides technical capabilities that support compliance efforts:

Regulatory Requirement	HJS Technical Support
Traceability	Cryptographically chained event records
Accountability	Explicit Actor field and signatures
Evidence preservation	Self-contained verifiable Receipts
Cross-system audit	Platform-agnostic Receipt format

5.2. Deployment Considerations

5.2.1. Multi-Platform AI Workflows

Organizations deploying AI across multiple platforms face operational challenges:

Observed Pattern (Healthcare):

* Primary screening on Cloud Platform A
* Specialist confirmation on Cloud Platform B
* Final reporting on on-premise system
* Regulatory audit requires reconciliation of three separate audit trails

Current Friction: Manual reconciliation of incompatible logs; no cryptographic verification of cross-platform handoffs.

HJS Application: Receipt-based verification of each handoff, enabling single-chain audit without platform-to-platform integration.

5.2.2. Autonomous Agent Delegation

Emerging use case: Agent-to-agent responsibility transfer.

Observed Pattern (Supply Chain):

* Agent A (manufacturer system) negotiates with Agent B (logistics system)

* Current practice: API calls with logged assertions
* Gap: No cryptographic proof of delegation for post-hoc audit

HJS Application: Delegate events with Receipts create portable, verifiable record of agent-to-agent responsibility transfer.

Note: This is an emerging use case with limited production volume.

5.3. Technical Evolution Considerations

5.3.1. Integration with Developing Standards

HJS is designed to integrate with standards that are currently in development:

Standard	Current Status	HJS Relationship
AIP (Agent Identity)	Draft	HJS events as audit log format
VAP (Provenance)	Draft	HJS as integrity layer implementation
SEAT (Session Security)	Draft	Session binding for replay protection
SCITT (Transparency)	Draft	Optional anchor_proof type

5.4. Adoption Evaluation Framework

Whether HJS provides value depends on observable outcomes:

Minimum traction indicators:

* Independent platform (non-Foundation) deploys to production with public documentation
* Regulatory evaluation publishes findings
* Additional platform vendors commit to integration timelines

5.5. Limitations and Uncertainties

5.5.1. Technical Limitations

HJS is designed for specific constraints. It does not address:

Limitation	Rationale	Current Workaround
No consensus mechanism	Avoids blockchain complexity	Platform-specific ordering for simultaneous events
No real-time sync	Offline verification priority	Eventual consistency via Receipt propagation
No built-in identity	Separation of concerns	Integration with DID/OIDC for actor verification

5.5.2. Market Uncertainties

Several factors could reduce or eliminate the need for HJS:

Factor	Observation	Impact on HJS
Platform consolidation	If AI market consolidates to 2-3 vendors	Cross-platform interoperability becomes less relevant
Regulatory divergence	If jurisdictions mandate incompatible formats	HJS may not satisfy all regional requirements
Alternative solutions	If existing standards (DID, SCITT) extend to cover judgment attribution	HJS becomes redundant

5.6. Success Criterion

HJS provides value when a regulator can verify an AI decision chain without needing real-time access to the original platforms' proprietary systems or databases, and with cryptographic certainty of the decision's authenticity and chain of custody.

6. Technical Specification

6.1. Judgment Event Model

6.1.1. Event Types

Event Type	Purpose	Modifies State	Has prev-hash	Portable Credential
Lifecycle Event	State transitions	Yes	Yes	Yes (via Receipt)
Verify Event	External attestation	No	No	Yes (independent proof)

6.1.2. Lifecycle Events

Lifecycle events form a cryptographically linked chain. Each event references its predecessor via prev_hash.

Three lifecycle primitives:

* Judge: Creates new judgment (genesis event)
* Delegate: Transfers responsibility (enables cross-system handoff)
* Terminate: Concludes judgment

6.1.3. Verify Events

Verify events exist outside the lifecycle chain. They reference a specific lifecycle event via event_id and provide third-party attestation without modifying judgment state.

6.2. Common Fields

Actor (lifecycle only)

URI identifying the entity responsible for the judgment.

Decision
: Cryptographic binding to decision content via decision_hash.

Authority Scope
: URI referencing the policy or framework governing the judgment.

Temporal Boundary
: Validity period via valid object with from and until timestamps.

Proof
: Cryptographic signature with algorithm identification.

6.3. State Machine

6.3.1. States

Symbol	Name	Description
A	Active	Judgment established; actor has control.
D	Delegated	Responsibility transferred; new actor has control.
P	Pending	Delegation initiated; awaiting acceptance.
T	Terminal	Judgment concluded; no further lifecycle operations.

6.3.2. State Transitions

From	To	Primitive	Condition
null	A	Judge	Genesis (prev_hash = null)
A	P	Delegate	Transfer initiated, awaiting acceptance
P	D	Delegate	Acceptance proof received
P	A	[Timeout]	Auto-revert if acceptance timeout
D	P	Delegate	Chained transfer initiated
A/D	T	Terminate	Normal conclusion
A/D/P	T	Terminate	Invalidation (with authority_proof)
A/D	T	Terminate	Timeout (system-triggered)

Invalid transitions (MUST be rejected):

* Any transition from Terminal state
* Judge with non-null prev_hash
* Delegate without acceptance proof (unless policy waives requirement)

6.3.3. Delegation and Acceptance Flow

```
      | 1. Create Delegate event        |
      |    State: A -> P (Pending)      |
      |                                 |
      | 2. Send Receipt + Request       |
      |-------------------------------->|
      |                                 |
      |                                 | 3. Verify Receipt
      |                                 | 4. Generate acceptance_proof
      |                                 |
      | 5. Return acceptance_proof      |
      |<--------------------------------|
      |                                 |
      | 6. Update state: P -> D         |
      |    (Delegation complete)         |
```

Timeout Handling: If acceptance_proof is not received within policy-defined window (RECOMMENDED: 30 seconds to 5 minutes), the delegation MUST auto-revert to Active state (A). The Pending event remains in the chain as a tombstone for audit purposes.

6.4. Data Format

6.4.1. JSON Representation

6.4.1.1. Lifecycle Event Schema

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "$id": "https://hjs.org/schema/lifecycle-event-v1.json",
  "title": "HJS Lifecycle Event",
  "type": "object",
  "required": ["hjs", "id", "actor", "decision_hash", "authority_scope",
               "valid", "state", "prev_hash", "primitive", "proof"],
  "properties": {
    "hjs": {"type": "string", "const": "1.0"},
    "id": {"type": "string", "format": "uuid"},
    "actor": {"type": "string", "format": "uri"},
    "decision_hash": {
      "type": "object",
      "required": ["hash", "alg"],
      "properties": {
        "hash": {"type": "string", "pattern": "^[a-f0-9]+$"},
        "alg": {"enum": ["sha256", "sha3-256", "blake2b", "hmac-sha256", "sm3"]}
      }
    },
    "authority_scope": {"type": "string", "format": "uri"},
    "valid": {
      "type": "object",
      "required": ["from", "until"],
      "properties": {
        "from": {"type": "integer", "minimum": 0},
        "until": {"type": "integer", "minimum": 0}
      }
    },
    "state": {"enum": ["A", "D", "P", "T"]},
    "prev_hash": {
      "oneOf": [
        {"type": "object", "required": ["hash", "alg"]},
        {"type": "null"}
      ]
    },
    "primitive": {
      "type": "object",
      "required": ["type"],
      "properties": {
        "type": {"enum": ["J", "D", "T"]},
```

```
          "payload": {
            "type": "object",
            "properties": {
              "to": {"type": "string", "format": "uri"},
              "acceptance_proof": {"type": "string"},
              "timeout_ms": {"type": "integer", "minimum": 0},
              "reason": {"enum": ["completed", "cancelled", "voided_by_authority", "expi
red", "acceptance_timeout"]},
              "authority_proof": {"type": "string"}
            }
          }
        }
      },
      "extensions": {"type": "object"},
      "proof": {
        "type": "object",
        "required": ["signature", "alg"],
        "properties": {
          "signature": {"type": "string"},
          "alg": {"type": "string"},
          "key_id": {"type": "string", "format": "uri"}
        }
      }
    }
  }
```

6.4.1.2. Verify Event Schema

```
  {
    "$schema": "http://json-schema.org/draft-07/schema#",
    "$id": "https://hjs.org/schema/verify-event-v1.json",
    "title": "HJS Verify Event",
    "type": "object",
    "required": ["hjs", "verify", "proof"],
    "properties": {
      "hjs": {"type": "string", "const": "1.0"},
      "verify": {
        "type": "object",
        "required": ["event_id", "method"],
        "properties": {
          "event_id": {"type": "string", "format": "uuid"},
          "method": {"type": "string"},
          "attestation": {
            "type": "object",
            "properties": {
              "format": {"enum": ["base64", "cbor", "jwt"]},
              "data": {"type": "string"},
              "alg": {"type": "string"}
            }
          },
          "verifier": {"type": "string", "format": "uri"},
          "timestamp": {"type": "integer"}
        }
      },
      "proof": {
        "type": "object",
        "required": ["signature", "alg"],
        "properties": {
          "signature": {"type": "string"},
          "alg": {"type": "string"},
          "key_id": {"type": "string", "format": "uri"}
        }
      }
    }
  }
```

6.4.2. Serialization

For signature generation and hash computation, events MUST be serialized using JSON Canonicalization Scheme (JCS) [RFC8785].

6.4.3. Hash Computation

```
// 1. Create a copy of the event
const hashInput = { ...event };

// 2. Remove the proof field
delete hashInput.proof;

// 3. Canonicalize using JCS
const canonical = jcs_canonicalize(hashInput);

// 4. Compute SHA-256 (or SM3 for China-compliant deployments)
const digest = sha256(canonical); // or sm3(canonical)

// 5. Format as object
const prev_hash = {
  "hash": hex_encode(digest),
  "alg": "sha256" // or "sm3"
};
```

6.5. HJS Receipt Format

6.5.1. Receipt Structure

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "$id": "https://hjs.org/schema/receipt-v1.json",
  "title": "HJS Receipt",
  "type": "object",
  "required": ["hjs_receipt", "version", "event_id", "event_hash",
               "chain_proof", "anchor_proof", "verification_mode"],
  "properties": {
    "hjs_receipt": {"type": "string", "const": "1.0"},
    "version": {"type": "string", "const": "1.0"},
    "event_id": {"type": "string", "format": "uuid"},
    "event_hash": {
      "type": "object",
      "required": ["hash", "alg"],
      "properties": {
        "hash": {"type": "string"},
        "alg": {"enum": ["sha256", "sha3-256", "blake2b", "sm3"]}
      }
    },
    "chain_proof": {
      "type": "object",
      "required": ["type", "anchor", "depth"],
      "properties": {
        "type": {"enum": ["linear", "merkle"]},
        "anchor": {
          "type": "object",
          "required": ["hash", "alg"],
          "properties": {
            "hash": {"type": "string"},
            "alg": {"type": "string"}
          }
        },
        "depth": {"type": "integer", "minimum": 0},
        "merkle_path": {
          "type": "array",
          "items": {"type": "string"}
        }
```

```
        }
      },
      "anchor_proof": {
        "type": "object",
        "required": ["type", "timestamp"],
        "properties": {
          "type": {
            "enum": ["memory", "tsp", "ots", "scitt", "platform"]
          },
          "timestamp": {"type": "integer"},
          "data": {"type": "string"},
          "uri": {"type": "string", "format": "uri"}
        }
      },
      "verification_mode": {"enum": ["platform", "open", "dual"]},
      "platform_attestation": {
        "type": "object",
        "properties": {
          "issuer": {"type": "string", "format": "uri"},
          "issued_at": {"type": "integer"},
          "signature": {"type": "string"},
          "alg": {"type": "string"},
          "key_id": {"type": "string", "format": "uri"}
        }
      },
      "extensions": {"type": "object"}
    }
  }
```

6.5.2. Receipt Verification

```
  function verifyReceipt(receipt, event) {
    // 1. Verify event hash matches
    if (computeHash(event) !== receipt.event_hash.hash) {
      return {valid: false, reason: "hash_mismatch"};
    }

    // 2. Verify chain integrity
    if (receipt.chain_proof.type === "linear") {
      // Verify linear chain
    } else if (receipt.chain_proof.type === "merkle") {
      // Verify Merkle path
    }

    // 3. Verify anchor proof
    switch(receipt.anchor_proof.type) {
      case "tsp": return verifyTSP(receipt.anchor_proof.data);
      case "ots": return verifyOTS(receipt.anchor_proof.data);
      case "scitt": return verifySCITT(receipt.anchor_proof.data);
      case "platform": return verifyPlatformSig(receipt.platform_attestation);
      default: return {valid: true}; // memory
    }
  }
```

6.6. Extensions

Extensions use reverse domain notation with version suffix:

```
  {
    "extensions": {
      "org.veritaschain.vap.integrity@v1": {
        "merkle_root": "sha256:...",
        "inclusion_proof": ["0x1234...", "0x5678..."]
      },
      "ietf.seat.binding@v1": {
        "session_id": "urn:uuid:...",
```

```
      "server_nonce": "...",
      "client_nonce": "..."
    }
  }
}
```

Guidelines:

- Extensions MUST NOT alter core protocol semantics
- Implementations SHOULD ignore unknown extensions
- Version suffix enables evolution

7. HJS Verification Layer

7.1. Verification Modes

Mode	Trust Root	Latency	Use Case
platform	HJS platform signature	<10ms	Real-time monitoring, high-trust environments
open	Receipt cryptographic proof	<1ms (local)	Cross-platform delegation, offline audit, regulatory verification
dual	Cross-confirmation	<100ms	High-value decisions, maximum assurance

7.2. HTTP API

7.2.1. Event Retrieval

```
GET /v1/events/{event_id}
```

Retrieve a lifecycle or verify event.

Response: LifecycleEvent | VerifyEvent

7.2.2. Chain Integrity Verification

```
GET /v1/chains/{chain_id}/integrity
```

Verify the integrity of a judgment lifecycle chain.

Query params:

- from: Starting event ID (optional)
- to: Ending event ID (optional)
- include_events: Boolean (default: false)

```
{
  "chain_id": "...",
  "valid": true,
  "length": 3,
  "genesis": "...",
  "latest": "...",
  "events": [...],
  "platform_attestation": {...}
}
```

7.2.3. Receipt Retrieval

```
GET /v1/events/{event_id}/receipt
```

Retrieve an HJS Receipt for offline verification.

Query params:

* mode: platform, open, or dual (default: open)
* anchor_type: tsp, ots, scitt, or memory (optional)

7.2.4. Verification

POST /v1/verify

Verify an event using specified mode.

Request:

```
{
  "event": {...},
  "receipt": {...},
  "mode_preference": "dual",
  "require_receipt": true
}
```

Response (success):

```
{
  "result": "valid",
  "mode_used": "dual",
  "checks": {
    "signature_valid": true,
    "chain_integrity": true,
    "anchor_verified": true,
    "platform_attestation": true
  },
  "verified_at": 1712345678
}
```

Response (disputed):

```
{
  "result": "disputed",
  "mode_used": "dual",
  "dispute_details": {
    "platform_result": "valid",
    "open_result": "invalid",
    "reason": "receipt_chain_invalid",
    "platform_checks": {
      "signature_valid": true,
      "chain_integrity": true
    },
    "open_checks": {
      "signature_valid": true,
      "chain_integrity": false,
      "chain_failure_reason": "prev_hash_mismatch"
    }
  },
  "verified_at": 1712345678
}
```

8. Ecosystem Integration

8.1. Complementary Systems

HJS provides judgment attribution that complements other systems in the accountability stack:

System	Their Contribution	HJS Contribution
VAP (Kamimura et al.)	Data/model provenance	Decision attribution
SEAT (Fossati et al.)	Session security	Decision binding
Arize	Real-time monitoring	Tamper-proof decision logs
WhyLabs	Data quality	Quality-event linkage
Fiddler	Governance enforcement	Enforcement audit trail

8.2. Complementary Relationship with SCITT

8.2.1. Core Differences

Dimension	HJS	SCITT
Design Goal	Cross-platform responsibility attribution	Global transparency logging
State Semantics	Stateful (A/D/P/T states)	Stateless, append-only
Verification Model	Offline-first, self-contained Receipt	Requires querying transparency logs
Trust Model	No pre-trust required between platforms	Trust in log operator honesty

8.2.2. Complementary Usage Patterns

Integration Pattern	HJS Role	SCITT Role
SCITT as Anchor Layer	Generate judgment events	Provide global transparency Receipt
HJS as Semantic Layer	Provide state machine and Receipt format	Optional log storage
Hybrid Mode	Core responsibility chain	Optional transparency enhancement

8.3. Alignment with VAP Framework

The Verifiable AI Provenance (VAP) Framework defines a layered architecture:

Layer	Description
Integrity	Tamper-evident recording of events
Provenance	Recording of decision inputs and contextual

	factors
Accountability	Attribution of responsibility and decision authority
Completeness	Guarantees against omission and selective disclosure

8.4. Integration with SEAT

The Session Evidence Attestation Transport (SEAT) protocol prevents replay attacks. SEAT and HJS provide complementary security:

* SEAT focuses on: "Is this communication session authentic?"
* HJS focuses on: "Is this decision properly attributed?"

8.5. Relationship to RATS Architecture

RATS Concept	HJS Mapping
Attester	The AI platform or agent generating a Lifecycle Event
Attestation Evidence	An HJS Lifecycle Event
Verifier	A relying party validating an HJS Receipt
Attestation Result	The HJS Receipt

8.6. Integration with Monitoring Platforms

Platform	Primary Focus	HJS Complement
Arize AI	Detecting model drift and performance issues	Recording which decisions triggered the drift
WhyLabs	Profiling data quality and anomalies	Linking quality issues to responsible decision-makers
Fiddler AI	Enforcing governance policies	Logging who made policy overrides

8.7. Relationship to AIP (Agent Identity Protocol)

Layer	AIP Focus	HJS Complement
Identity	Who is the agent?	Records which agent made which decision
Governance	What policies apply?	Records which policies were invoked
Enforcement	How are policies enforced?	Records enforcement actions
Evidence	(AIP defers this)	Tamper-evident decision logs

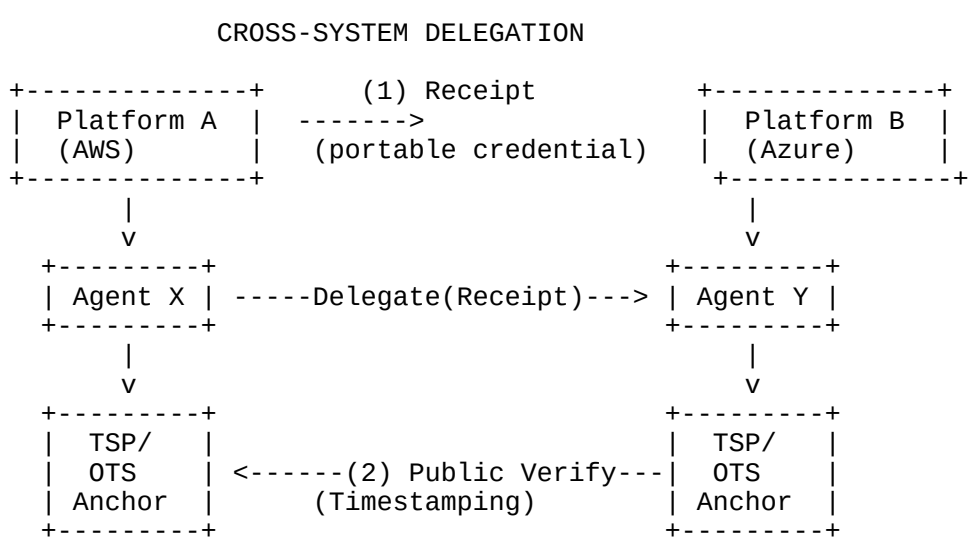
8.8. HJS as Inter-Agent Communication Protocol

8.8.1. The Cross-System Delegation Problem

Consider a multi-platform AI workflow:

* Platform A: Medical AI system makes initial diagnosis
* Need: Transfer to specialized system on Platform B for second opinion
* Challenge: Platform B must accept responsibility transfer without trusting Platform A
* Regulatory Requirement: External auditor must verify complete chain without platform access

8.8.2. Cross-System Delegation Flow



8.8.3. Receipt as Universal Credential

Property	Mechanism	Benefit
Self-contained	Includes all proofs in single JSON object	No external dependencies for verification
Tamper-evident	Hash chain + cryptographic signatures	Any modification invalidates the proof
Time-bound	TSP/OTS/SCITT anchoring to public ledgers	Independent timestamp verification
Privacy-preserving	Decision content hashed, metadata selective	Reveal only necessary information
Extensible	Standard extension mechanism	Carry partner protocol data

8.9. What HJS Does NOT Do

HJS intentionally does NOT provide:

* Data provenance -> VAP, SCITT
* Session security -> SEAT, TLS

* Real-time monitoring -> Arize, WhyLabs
* Governance policy engines -> Fiddler, organizational systems
* Model explainability -> SHAP, LIME
* Identity verification -> DID, SPIFFE, AIP
* Regulatory compliance judgments -> Legal and compliance professionals

9. Security Considerations

9.1. Signature Validation

All lifecycle transitions MUST be cryptographically signed by the responsible actor.

9.2. Chain Integrity

Implementations MUST verify:

* Each prev_hash matches the previous event's hash
* No forks or gaps in the chain
* Genesis event has prev_hash: null

9.3. Replay Attacks and Connection Binding

To prevent relay attacks, implementations SHOULD bind HJS events to the connection context using mechanisms such as those defined in [I-D.fossati-seat-expat]. Proof that is not bound to the connection may be susceptible to replay across different sessions.

Implementations SHOULD:

* Use UUIDv7 for unique event IDs
* Validate timestamps against acceptable skew
* Bind events to domain-specific context
* Use SEAT session binding for connection-oriented protocols

9.4. Delegation Security

To prevent unauthorized delegation:

* Implementations SHOULD require acceptance_proof in Delegate payload
* Acceptance MUST be signed by the receiving actor
* Timeout and auto-terminate if acceptance not received within policy window

9.5. Time Source Security

Implementations MUST:

* Use at least two independent time sources (e.g., NTP + TSP/OTS)
* Validate time skew < 1000ms between sources
* Reject future timestamps (> now + 60s)
* Reject timestamps significantly in the past (< now - 86400s, i.e., 24 hours)
* Include time attestations in Receipts when available

9.6. UUIDv7 Clock Dependency

Implementations generating UUIDv7 for event id fields MUST ensure that the system clock is synchronized using a secure protocol (e.g., NTP with authentication) and is protected from unauthorized modification. In high-security environments where clock manipulation is a concern, implementations SHOULD consider using UUIDv4 instead, trading off time-sortability (a key feature of UUIDv7) for reduced reliance on clock integrity. The choice of UUID version is an implementation decision; the protocol supports both.

9.7. Trust Model Summary

HJS provides cryptographic evidence of who said what when. The verification of whether they *should* have said it is addressed by governance systems and partner platforms.

9.8. Post-Quantum Cryptography Migration

HJS is designed for long-term verifiability—a judgment Receipt created today may need verification in 10, 20, or 30 years. This requires consideration of quantum computing threats to current cryptographic algorithms.

9.8.1. Risk Assessment

Algorithm	Classical Security	Quantum Threat	Estimated Risk Timeline
Ed25519	High	Shor’s algorithm	2030-2035
SHA-256	High	Grover’s algorithm	Requires hash length doubling
SM2/SM3	High	Same as above	Same as above

9.8.2. Algorithm-Agnostic Design

HJS data structures are designed for PQC migration:

```
{
  "proof": {
    "alg": "ed25519",        // Extensible to multiple algorithms
    "signature": "base64..."
  }
}
```

9.8.3. Dual Signatures (High-Security Scenarios)

For scenarios requiring long-term assurance, implementations can use both classical and PQC signatures:

```
{
  "proof": [
    {
      "alg": "ed25519",
      "signature": "base64-classic-sig"
    },
    {
      "alg": "ml-dsa-65",    // ML-DSA (FIPS 204)
      "signature": "base64-pqc-sig"
    }
  ]
}
```

9.8.4. Migration Timeline (March 2026)

Phase	Timeframe	Recommended Actions
Preparation	Current (2026)	Implement optional PQC support alongside classical signatures
Transition	2026-2028	New systems enable PQC by default; critical systems require dual

		signatures
Completion	2028+	Classical signatures marked as "legacy"; PQC becomes primary

9.8.5. Long-Term Receipt Verification

For Receipts that must remain verifiable for decades:

1. Periodic Re-anchoring: Re-anchor Receipt hashes to new timestamp services every 5-10 years
2. Signature Refresh: Re-sign old Receipts with newer algorithms (preserving original signatures)
3. Context Preservation: Save verification context (CA certificates, algorithm specifications)

```
{
  "hjs_receipt": "1.0",
  "event_id": "2026-event",
  "proof_history": [
    {
      "alg": "ed25519",
      "signature": "...",
      "valid_from": "2026-01-01"
    },
    {
      "alg": "ml-dsa-65",
      "signature": "...",
      "valid_from": "2028-01-01",
      "re_signed_by": "original-platform"
    }
  ]
}
```

10. Privacy Considerations

10.1. Decision Content Privacy

Use HMAC or salted hashing for low-entropy decisions to prevent rainbow table attacks.

10.2. Metadata Leakage

Event fields may reveal sensitive information. Implementations SHOULD:

* Use pseudonymous identifiers for actor in public contexts
* Encrypt sensitive extension fields
* Minimize metadata disclosure

10.3. Data Minimization

Implementations SHOULD only collect metadata essential for accountability, deferring domain-specific requirements to extensions.

10.4. GDPR and the "Right to be Forgotten"

HJS events are designed to be immutable and tamper-evident, which creates tension with GDPR Article 17 ("right to erasure"). This specification does not resolve this tension but notes current practices:

Approach	Description	Trade-off
Cryptographic deletion	Key destruction rendering events undecryptable	Irreversible; may violate audit

		requirements
Redaction markers	Tombstone events marking data as "erased"	Maintains chain integrity; visible that data existed
Off-chain data	Store sensitive data externally, reference via hash	HJS events remain; external data can be deleted
Legal basis	Rely on "legitimate interest" or "legal obligation"	Requires legal assessment; not technical solution

11. IANA Considerations

11.1. Media Types

Type name: application
Subtype name: hjs+json
Required parameters: None
Optional parameters:
    version (default: "1.0")
    type (values: "lifecycle", "verify")
Encoding considerations: Binary
Security considerations: See Section 9
Published specification: This document
Contact: Yuqiang Wang <signal@humanjudgment.org>
Intended usage: COMMON
Change controller: IETF

Type name: application
Subtype name: hjs-receipt+json
Required parameters: None
Optional parameters:
    version (default: "1.0")
    mode (values: "full", "existence", "selective")
Encoding considerations: Binary
Security considerations: See Section 9
Published specification: This document
Contact: Yuqiang Wang <signal@humanjudgment.org>
Intended usage: COMMON
Change controller: IETF

11.2. Well-Known URI

URI suffix: hjs
Change controller: IETF

12. Normative References

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

[RFC3161] Adams, C., Cain, P., Pinkas, D., and R. Zuccherato, "Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP)", RFC 3161, DOI 10.17487/RFC3161, August 2001, <https://www.rfc-editor.org/info/rfc3161>.

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

[RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <https://www.rfc-editor.org/info/rfc8259>.

[RFC8785] Rundgren, A., Jordan, B., and S. Erdtman, "JSON Canonicalization Scheme (JCS)", RFC 8785, DOI 10.17487/RFC8785, June 2020, <https://www.rfc-editor.org/info/rfc8785>.

[RFC9334] Birkholz, H., Thaler, D., Richardson, M., Smith, N., and W. Pan, "Remote ATtestation procedureS (RATS) Architecture", RFC 9334, DOI 10.17487/RFC9334, January 2023, <https://www.rfc-editor.org/info/rfc9334>.

[RFC9562] Davis, K., Peabody, B., and M. Leach, "Universally Unique IDentifiers (UUIDs)", RFC 9562, DOI 10.17487/RFC9562, May 2024, <https://www.rfc-editor.org/info/rfc9562>.

13. Informative References

[I-D.fossati-seat-expat]
Fossati, T. and H. Tschofenig, "Session Evidence Attestation Transport (SEAT) and Exported Authenticators", Work in Progress, Internet-Draft, draft-fossati-seat-expat-00, 27 February 2026, <https://datatracker.ietf.org/doc/html/draft-fossati-seat-expat-00>.

[I-D.ietf-scitt-architecture]
Birkholz, H., Steele, O., Hoyland, J., Richardson, M., and S. Hunt, "An Architecture for Trustworthy Digital Supply Chains", Work in Progress, Internet-Draft, draft-ietf-scitt-architecture-04, 27 January 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-scitt-architecture-04>.

[I-D.kamimura-vap-framework]
Kamimura, T., "Verifiable AI Provenance (VAP) Framework", Work in Progress, Internet-Draft, draft-kamimura-vap-framework-00, 23 February 2026, <https://datatracker.ietf.org/doc/html/draft-kamimura-vap-framework-00>.

Acknowledgments

HJS was developed with input from protocol authors, platform teams, and industry experts who each contribute essential capabilities to AI accountability:

Protocol Design Partners

Tokachi Kamimura and the VAP team — Their work on provenance layers clarified the architectural boundary between integrity, provenance, and accountability.

Thomas Fossati and the SEAT team — Their analysis of relay attacks informed the session binding mechanisms that protect HJS events in transit.

James (yungcero) and the Agent Identity Protocol (AIP) team — Their work on agent identity and governance layers provided a concrete context for HJS’s evidence layer.

Platform Integration Partners

Arize AI — Production requirements for real-time verification helped validate the dual verification mode.

Fiddler AI — Governance enforcement use cases shaped the Terminate reason codes and authority validation logic.

WhyLabs — Requirements for linking data quality events to decision attribution informed the extension design.

Industry Experts

Wang Aijun (DMSC) — Professional guidance and support that contributed to the technical development of HJS.

Community Contributors

Usama — Feedback on relay attacks and security considerations prompted stronger alignment with SEAT.

Arnaud Taddei — Insights on SG17/RATS work helped clarify HJS's relationship to broader attestation frameworks.

Appendix A. Reference Implementation

A Rust-based reference implementation of the HJS protocol is planned and will be released as open source on GitHub. The repository URL will be provided in a future revision of this document.

Appendix B. Dispute Handling Decision Framework

B.1. Overview

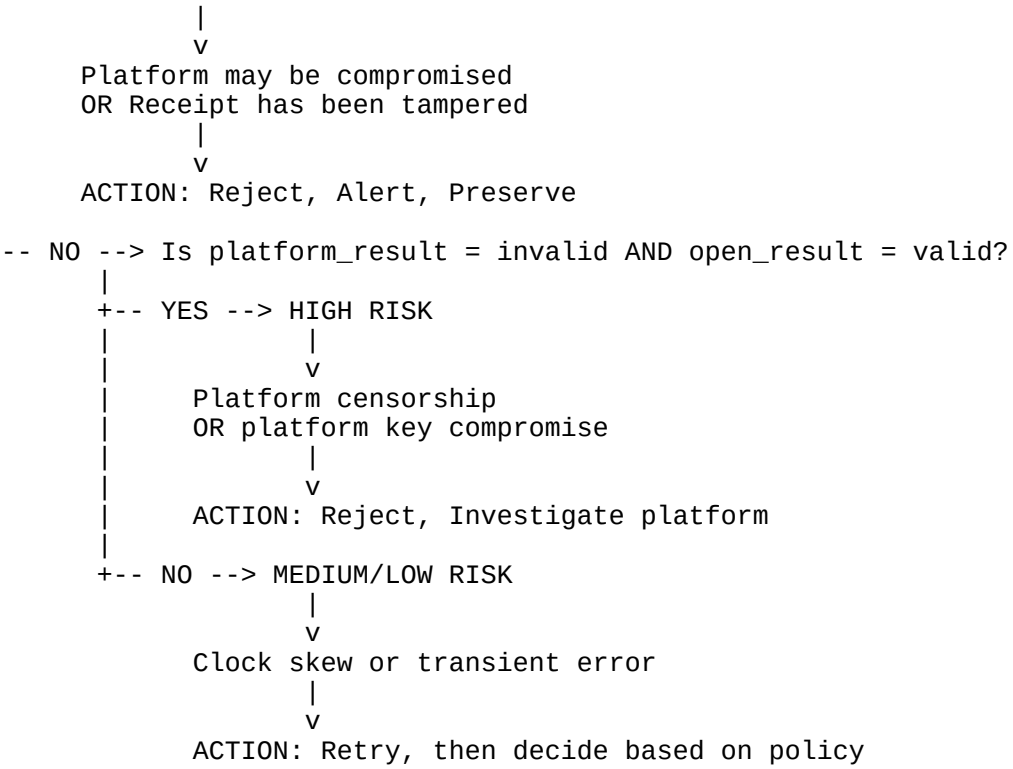
When dual mode verification returns disputed, the relying party must make a trust decision. This appendix provides a decision framework, not a normative protocol requirement.

B.2. Risk Assessment Matrix

Scenario	Platform Result	Open Result	Risk Level	Recommended Action
1	valid	invalid	Critical	Reject event; alert security team; preserve evidence
2	invalid	valid	High	Reject event; platform may be compromised or censored
3	valid	valid (mismatch)	Medium	Clock skew or implementation bug; retry with fresh data
4	timeout	valid	Low-Medium	Accept open result if platform unreachable; log anomaly
5	timeout	invalid	High	Reject event; insufficient verification

B.3. Decision Tree

```
DISPUTE RECEIVED
       |
       v
Is platform_result = valid AND open_result = invalid?
       |
       +-- YES --> CRITICAL RISK
```



B.4. Implementation Recommendations

Logging: All disputes MUST be logged with:

* Full Receipt content
* Platform attestation response (if any)
* Local verification details
* Timestamp and context

Alerting: Critical and High risk disputes SHOULD trigger immediate alerts to security operations.

Evidence Preservation: Dispute data may be required for forensic analysis. Implementations SHOULD retain dispute records for at least 90 days or per regulatory requirement.

Appendix C. Implementation Risks and Mitigations

C.1. Asynchronous Delegation Risks

Risk: Delegate operation requires acceptance_proof from receiving platform, creating availability dependency.

Mitigation:

* Pending state with timeout (Section 6.3.3)
* Idempotent acceptance handling for retries
* Clear error semantics for timeout vs. rejection

C.2. Clock Synchronization Risks

Risk: UUIDv7 and timestamp validation depend on system clock accuracy.

Mitigation:

* Multi-source time validation (Section 9.5)
* UUIDv4 fallback option (Section 9.6)
* Tolerance windows for clock skew

C.3. Platform Compromise Detection

Risk: Dual mode detects compromise but creates false positives under benign conditions (clock skew, network issues).

Mitigation:

* Retry logic for transient failures
* Dispute classification (Appendix B)
* Graduated response rather than hard failure

C.4. Adoption Dependencies

Risk: HJS value depends on multi-platform adoption. Single-platform deployment provides limited benefit.

Mitigation:

* Reference implementation availability
* Clear integration patterns with existing platforms

C.5. Alternative Deployment Paths

If broad IETF standardization does not achieve traction, HJS may provide value through:

Path	Condition	Value Proposition
Vertical solution	Single industry (healthcare/finance) adopts	Deep integration with sector-specific compliance requirements
Open source project	Apache 2.0 governance, community-driven	Rapid iteration without standardization overhead
Regulatory technology	Direct engagement with specific regulators	Compliance tooling rather than universal standard

Author's Address

Yuqiang Wang
HUMAN JUDGMENT SYSTEMS FOUNDATION LTD.
Email: signal@humanjudgment.org
GitHub: https://github.com/schchit