Internet Engineering Task Force
Internet-Draft
Intended status: Informational
Expires: 26 September 2026

Helixar
Helixar Limited

March 2026

Human Delegation Provenance Protocol (HDP): Cryptographic Chain-of-Custody for Agentic AI Systems

draft-helixar-hdp-agentic-delegation-00

Abstract

Agentic AI systems operate on behalf of human principals, often delegating tasks through multi-step chains of AI agents. There is currently no standard mechanism to record who authorized an agent to act, under what scope, and through what chain of delegation , in a way that can be verified offline, without a central registry, and without third-party trust anchors.

This document specifies the Human Delegation Provenance Protocol (HDP) version 0.1, a lightweight token-based protocol that captures, structures, cryptographically signs, and verifies human delegation context in agentic AI systems. An HDP token binds a human authorization event to a session, records each agent’s delegation action as a signed hop in an append-only chain, and enables any participant to verify the full provenance record using only the issuer’s Ed25519 public key and the current session identifier. Verification is fully offline. No registry lookup, no network call, and no third-party trust anchor is required.

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 2 September 2026.

Table of Contents

1. Introduction

Autonomous AI agents are increasingly used to execute consequential actions: sending emails, modifying files, running code, calling APIs, and transacting on behalf of users. When a human authorizes an orchestrator agent, which in turn delegates to sub-agents, which further delegate to tool-execution agents, the originating human authorization becomes disconnected from the terminal action. There is no standard record of the authorization chain.

This gap creates accountability, auditability, and safety problems:

* Downstream agents cannot verify that the action they are being asked to perform was actually authorized by a human.

* Post-hoc audits cannot reconstruct who approved what, and when.

* Prompt injection attacks , where malicious content in the environment instructs an agent to act , cannot be distinguished from legitimate human delegation.

HDP addresses this by defining a token that:

* Records the human principal, their declared scope, and the session binding at issuance.

* Accumulates a cryptographically signed hop record for each agent that handles the token.

* Allows any recipient to verify the entire chain , root signature plus all hop signatures , using only the issuer's Ed25519 public key and the session identifier.

1.1. Motivation

The need for agentic delegation provenance is not hypothetical. Production deployments of AI orchestration systems (LangChain, AutoGPT, CrewAI, and similar frameworks) today pass natural language task descriptions between agents with no cryptographic binding to the original human authorization. The operational risk compounds as models become more capable and agents are granted access to higher-consequence tools.

A provenance token that travels alongside the task , tamper-evident, offline-verifiable, and scoped to what the human actually approved , provides the foundation for auditable, accountable agentic systems.

1.2. Design Goals

HDP is designed with the following goals in order of priority:

1. *Offline verifiability.* Verification MUST require only a public key and session ID. No network call, registry lookup, or third-party endpoint is required.

2. *Self-sovereignty.* Any organization MUST be able to issue and verify HDP tokens without registering with a central authority or anchoring to a third-party key.

3. *Tamper evidence.* Any modification to a token , in its header, principal, scope, or any hop , MUST be detectable by the verification pipeline.

4. *Minimal footprint.* The protocol MUST be implementable in any language with Ed25519 and JSON support. No mandatory infrastructure beyond key management is required.

5. *Privacy by design.* Principal identity fields MUST be separable from the audit-relevant parts of the token, so tokens can be transmitted to agents without exposing PII.

1.3. Relationship to IPP (draft-haberkamp-ipp-00)

The Intent Provenance Protocol [I-D.haberkamp-ipp] addresses the same problem space. HDP and IPP share the use of Ed25519 signatures and append-only provenance chains but make different architectural trade-offs, which are detailed in Section 12. The two protocols are not interoperable. HDP is offered as a distinct design point, not a revision of IPP.

The full HDP protocol specification is available at [HDP-SPEC]. A TypeScript reference implementation is available at [HDP-IMPL].

2. Conventions and Definitions

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

Issuer: The system or person that creates and signs an HDP token on behalf of a human principal.

Principal: The human who authorized the agentic task. Represented in the token's principal object.

Agent: Any AI system, model, or automated process that receives and acts upon an HDP token.

Hop: A single delegation event, recorded as a signed entry in the token's chain array.

Root signature: The Ed25519 signature over the token's header, principal, and scope, computed by the issuer at token creation time.

Hop signature: The Ed25519 signature computed by an extending agent over the cumulative chain state at the time of extension.

Session: A logical unit of work identified by a session_id string, established between the issuer and the agent framework before the token is issued.

3. Token Structure

An HDP token is a JSON object with six top-level fields. The token MUST conform to the following structure. All integer timestamps are Unix milliseconds (milliseconds since 1970-01-01T00:00:00Z).

```
{
  "hdp"       : "0.1",          // protocol version
  "header"    : { ... },        // session binding + lifecycle
  "principal" : { ... },        // authorizing human
  "scope"     : { ... },        // authorized intent + constraints
  "chain"     : [ ... ],        // delegation hops (append-only)
  "signature" : { ... }         // root Ed25519 signature
}
```

Figure 1: HDP Token Top-Level Structure

3.1. Header

The header object carries token lifecycle and session binding fields.

```
{
  "token_id"        : "550e8400-e29b-41d4-a716-446655440000",
  "issued_at"       : 1711483200000,
  "expires_at"      : 1711569600000,
  "session_id"      : "sess-20260326-abc123",
  "version"         : "0.1",
  "parent_token_id" : "..."     // OPTIONAL: re-authorization linkage
}
```

token_id: REQUIRED. UUID v4. Unique identifier for this token.

issued_at: REQUIRED. Unix milliseconds. Time of issuance.

expires_at: REQUIRED. Unix milliseconds. Token MUST NOT be accepted after this time. Default lifetime is 24 hours.

session_id: REQUIRED. Opaque string. Established out-of-band between issuer and agent framework before token issuance. Provides replay defense: a token is only valid within the session for which it was issued.

version: REQUIRED. MUST equal the value of the top-level hdp field.

parent_token_id: OPTIONAL. If present, identifies the token this token supersedes in a re-authorization chain. See Section 6.

3.2. Principal

The principal object identifies the authorizing human. It MUST contain id and id_type. All other fields are OPTIONAL.

```
{
  "id"             : "usr_alice_opaque",
  "id_type"        : "opaque",
  "display_name"   : "Alice Chen",
  "poh_credential" : "...",
  "metadata"       : {}
}
```

The id_type field MUST be one of the following registered values, or a custom string prefixed with x-:

* opaque: Application-defined identifier. No resolution semantics are implied.

* email: RFC 5321 email address.

* uuid: RFC 4122 UUID.

* did: W3C Decentralized Identifier [W3C.DID]. DID resolution is application- defined and not required by this protocol.

* poh: A Proof-of-Humanity credential identifier. Verification semantics are application-defined; see Section 9.3.

HDP does not mandate any specific identity model. The did id_type is available for deployments with existing DID infrastructure; it is not required.

3.3. Scope

The scope object records what the human authorized. It is signed as part of the root signature and MUST NOT be modified after issuance.

```
{
  "intent"               : "Analyze Q1 sales data and produce a report.",
  "authorized_tools"     : ["database_read", "file_write"],
  "authorized_resources" : ["db://sales/q1-2026"],
  "data_classification"  : "confidential",
  "network_egress"       : false,
  "persistence"          : true,
  "max_hops"             : 3
}
```

intent: REQUIRED. Natural language description of the authorized task. Free-form string. This is the human-readable authorization statement.

authorized_tools: OPTIONAL. Array of tool identifiers the principal has explicitly authorized. Enforcement is application-defined.

authorized_resources: OPTIONAL. Array of resource identifiers (URIs, paths, etc.) the principal has authorized access to.

data_classification: REQUIRED. One of: public, internal, confidential, restricted. Expresses the sensitivity level of data the agent is authorized to access.

network_egress: REQUIRED. Boolean. Whether the agent is authorized to make outbound network requests.

persistence: REQUIRED. Boolean. Whether the agent is authorized to write persistent state.

max_hops: OPTIONAL. Positive integer. Maximum number of delegation hops permitted. Verification MUST reject tokens whose chain length exceeds this value.

HDP does not mandate a central taxonomy for intent, authorized_tools, or authorized_resources. These are self-described by the issuer. Semantic validation of agent actions against declared scope is an application-layer concern.

3.4. Chain

The chain array is append-only. Each element records a single delegation event (hop). The array is empty at issuance and grows as the token passes through agents. Agents MUST NOT remove or modify existing entries.

```
{
  "seq"               : 1,
  "agent_id"          : "orchestrator-v2",
  "agent_type"        : "orchestrator",
  "agent_fingerprint" : "sha256:abc123...",
  "timestamp"         : 1711483260000,
  "action_summary"    : "Decompose analysis task; delegate to sub-agents.",
  "parent_hop"        : 0,
  "hop_signature"     : "<base64url-encoded Ed25519 signature>"
}
```

seq: REQUIRED. Positive integer. Sequential index, starting at 1. MUST be exactly one greater than the previous hop's seq. Gaps in sequence are a protocol violation.

agent_id: REQUIRED. Identifier of the agent adding this hop.

agent_type: REQUIRED. One of: orchestrator, sub-agent, tool-executor, custom.

agent_fingerprint: OPTIONAL. Model or binary fingerprint for the acting agent.

timestamp: REQUIRED. Unix milliseconds. Time of hop extension.

action_summary: REQUIRED. Human-readable description of the action this agent intends to take.

parent_hop: REQUIRED. Non-negative integer. Index of the hop that

triggered this delegation, where 0 indicates the root (human) authorization.

hop_signature: REQUIRED. Base64url-encoded Ed25519 signature. See Section 4.2. Absence is a protocol violation per Rule 6 of Section 4.3.

3.5. Signature

The signature object carries the root signature computed by the issuer.

```
{
  "kid"   : "alice-signing-key-v1",
  "alg"   : "Ed25519",
  "value" : "<base64url-encoded Ed25519 signature over canonical JSON>"
}
```

The alg field MUST be Ed25519 for HDP v0.1. The kid field SHOULD be used by verifiers to identify the correct public key when multiple keys are in circulation.

4. Cryptographic Signing

4.1. Root Signature

The root signature is computed by the issuer at token creation time. It covers the token’s header, principal, and scope , the fields that constitute the human authorization event.

The signing procedure is:

1. Construct the unsigned token object containing the hdp, header, principal, scope, and chain (empty array at issuance) fields.

2. Serialize the object to canonical JSON using RFC 8785 [RFC8785] (JSON Canonicalization Scheme). This ensures deterministic byte representation across implementations and platforms.

3. Compute the Ed25519 [RFC8032] signature over the canonical JSON bytes using the issuer’s private key.

4. Encode the signature bytes as base64url [RFC4648] (no padding).

5. Attach the signature object (kid, alg, value) to the token.

The signature field itself MUST NOT be included in the canonical JSON payload before signing. The signed payload is deterministically recoverable by stripping the signature field from the complete token and re-serializing with RFC 8785.

4.2. Hop Signature

Each hop MUST carry a hop_signature. This signature binds the new hop record to the entire accumulated delegation history and to the root signature, making retroactive chain modification detectable.

The hop signing procedure is:

1. Construct the new hop record (all fields except hop_signature).

2. Build the signing payload as a JSON array: [hop_1, hop_2, ..., hop_(n-1), new_hop_unsigned] where hop_1 through hop_(n-1) are the previously signed hops (WITH their hop_signature fields) and new_hop_unsigned is the new hop record WITHOUT its hop_signature.

3. Prepend the root signature value (base64url string) to the array as its first element: [root_sig_value, hop_1, ..., new_hop_unsigned]. This chains the hop signature to the root.

4. Serialize the array to canonical JSON per RFC 8785.

5. Compute the Ed25519 signature over the canonical JSON bytes using the extending agent’s private key.

6. Encode as base64url and attach as the hop_signature field on the new hop record.

7. Append the signed hop to the token’s chain array.

The asymmetry between previously-signed hops (WITH hop_signature) and the new hop (WITHOUT hop_signature) in step 2 is intentional and critical. The verifier MUST reconstruct this exact payload structure when verifying each hop. See Section 5.

4.3. Chain Integrity Rules

The following rules govern chain construction and MUST be enforced by both extenders and verifiers:

1. Hop seq values MUST start at 1 and increment by exactly 1. No gaps are permitted.

2. Existing hop records MUST NOT be modified or removed.

3. A hop's parent_hop MUST reference a valid prior hop index (0 for the root human authorization, or the seq value of a prior hop).

4. If scope.max_hops is set, the chain length MUST NOT exceed it. A token with a full chain MUST NOT be extended.

5. Each hop's timestamp SHOULD be monotonically non-decreasing.

6. The hop_signature field MUST be present on every hop. A hop without a hop_signature is a protocol violation and MUST cause verification to fail.

5. Verification Pipeline

A verifier MUST execute the following seven steps in order. A failure at any step MUST cause immediate rejection with an appropriate error. The verifier MUST NOT proceed to subsequent steps after a failure.

1. *Version check.* The hdp field MUST contain a recognized protocol version string. For this specification, the only recognized value is "0.1".

2. *Expiry check.* The token's header.expires_at MUST be strictly greater than the current time. Expired tokens MUST be rejected.

3. *Root signature verification.* Reconstruct the canonical JSON payload by stripping the signature field and serializing the remaining token object per RFC 8785. Verify the Ed25519 signature in signature.value against this payload using the issuer's public key. A failure indicates tampering with the header, principal, or scope.

4. *Hop sequence integrity.* For each hop in chain, verify that hop.seq == (index + 1). Any gap or duplication MUST cause rejection.

5. *Hop signature verification.* For each hop at index i:

    a. Verify that hop_signature is present. Absence MUST cause rejection.

    b. Reconstruct the signing payload as described in Section 4.2, using the hops at indices 0...(i-1) with their signatures, plus the hop at index i without its hop_signature, prepended by the root signature value.

   c. Serialize the payload per RFC 8785 and verify the hop_signature against the issuer's public key (the same key used for the root signature in HDP v0.1).

6. *max_hops check.* If scope.max_hops is defined, the length of chain MUST NOT exceed it.

7. *Session binding check.* The token's header.session_id MUST exactly match the session_id provided by the verifying application. This prevents token replay across sessions. See Section 10.4.

An optional eighth step MAY be performed if the application has registered a Proof-of-Humanity verifier: if principal.poh_credential is present and a verifier callback is configured, the credential MUST be validated by that callback. See Section 9.3.

Verification is fully offline. Steps 1 through 7 require only the issuer's Ed25519 public key and the current session identifier. No network call, registry lookup, or third-party contact is required at any step.

6. Re-Authorization

Long-running or streaming sessions may exhaust the max_hops limit, require scope expansion, or encounter situations where a high-risk action warrants fresh human confirmation. In these cases, the issuer (acting on behalf of the human principal) issues a new token that supersedes the original.

Re-authorization is indicated by setting header.parent_token_id to the token_id of the token being superseded. This field MUST be set before computing the root signature, so the parentage link is cryptographically covered by the new token's root signature.

A re-authorized token:

* Has a new token_id, issued_at, and expires_at.

* Inherits session_id, principal, and scope from the original unless explicitly overridden.

* Starts with an empty chain (delegation count resets).

* Records parent_token_id pointing to the original, creating an auditable lineage of scope evolution.

Verifiers that require re-authorization chain traversal SHOULD retain all tokens in a session and verify the full parent_token_id linkage.

7. Multi-Principal Delegation

HDP v0.1 supports one principal per token. Joint authorization by multiple humans is achieved by sequential chaining: Human A issues token T1; Human B issues token T2 with parent_token_id equal to T1's token_id. Each token is independently signed with its issuer's key.

To verify a multi-principal chain, the verifier MUST:

1. Verify each token individually against its issuer's public key using the standard 7-step pipeline.

2. Verify that T[i].header.parent_token_id == T[i-1].header.token_id for all i > 0.

3. Verify that all tokens in the chain share the same session_id.

This pattern provides joint authorization auditably without requiring a threshold signature scheme. Each principal's authorization is a distinct signed artifact. A future version of HDP (v0.2) is planned to introduce simultaneous multi- signature primitives using threshold signature schemes.

8. Transport

8.1. HTTP Header: X-HDP-Token

HDP tokens MAY be transmitted in HTTP requests and responses using the X-HDP-Token header. The header value is the base64url encoding (RFC 4648, no padding) of the UTF-8 JSON serialization of the complete token object.

```
POST /api/task HTTP/1.1
Host: agent.example.com
X-HDP-Token: eyJoZHAiOiIwLjEiLCJoZWFkZXIiOnsi...
Content-Type: application/json
```

Figure 2: X-HDP-Token HTTP Header Example

Implementations MUST NOT include tokens in URL query parameters, as this exposes sensitive data in server logs and browser history.

8.2. Token by Reference: X-HDP-Token-Ref

When token size is a concern (e.g., large chains), the token MAY be stored server-side and referenced by its token_id using the X-HDP-Token-Ref header.

```
POST /api/task HTTP/1.1
Host: agent.example.com
X-HDP-Token-Ref: 550e8400-e29b-41d4-a716-446655440000
```

Figure 3: X-HDP-Token-Ref HTTP Header Example

Implementations using token-by-reference MUST secure the token store and use transport-layer security (TLS) for all reference resolution.

8.3. Key Distribution: Well-Known Endpoint

Issuers that wish to publish their Ed25519 public keys for automated discovery SHOULD serve a JSON document at /.well-known/hdp-keys.json with the following structure:

```
{
  "keys": [
    {
      "kid" : "alice-signing-key-v1",
      "alg" : "Ed25519",
      "pub" : "<base64url-encoded 32-byte Ed25519 public key>"
    }
  ]
}
```

This format is intentionally minimal. Implementations MAY extend it with additional metadata. The alg field MUST be "Ed25519" for HDP v0.1 keys. Consumers MUST reject entries with unrecognized alg values. Consumers MUST validate that the decoded public key is exactly 32 bytes.

9. Privacy Considerations

9.1. Minimum-Disclosure Principal Fields

The principal object may contain PII (email address, display name). Issuers SHOULD apply the principle of minimum disclosure when constructing tokens that will traverse multiple agents. Specifically:

* Use id_type: "opaque" with an application-internal identifier rather than embedding the user's email address in tokens that will be sent to third-party agents.

* Omit display_name when the receiving agent does not require a human-readable identity.

The token structure separates the identity fields (principal) from the audit-relevant fields (header, scope, chain). Implementations MAY strip the principal object when forwarding tokens to agents that do not require principal identity, while preserving the integrity of the signature chain. Note that stripping principal invalidates the root signature; stripped tokens MUST be clearly marked as audit-only records and MUST NOT be presented for signature verification.

9.2. Data Retention and the Right to Erasure

HDP tokens may constitute personal data under applicable privacy regulations (e.g., GDPR Article 4(1)) when the principal.id or principal.display_name fields contain directly or indirectly identifying information.

Implementations SHOULD:

* Store tokens with explicit retention periods derived from header.expires_at.

* Provide deletion mechanisms that remove stored tokens upon erasure requests.

* Use opaque identifiers in principal.id where possible, maintaining a separate mapping that can be destroyed independently of the token audit log.

9.3. Proof of Humanity

The optional principal.poh_credential field MAY carry a credential attesting that the principal is a human (e.g., a Worldcoin World ID proof, a CAPTCHA session token, or a biometric attestation identifier). The HDP protocol does not define the semantics of this field; verification is entirely application-defined.

When a PoH verifier is configured, the verification pipeline MUST validate the credential as the final step (after session binding) and MUST reject the token if validation fails. The verifier callback SHOULD be idempotent and SHOULD NOT have side effects.

10. Security Considerations

10.1. Threat Model

HDP is designed to provide provenance and tamper evidence, not runtime enforcement. An agent that exceeds its declared scope is still a bad actor , HDP creates an evidence trail, not a capability boundary. Applications requiring runtime enforcement MUST implement it at the application layer using the HDP token as audit input.

10.2. Token Forgery

A forged token , one whose header, principal, or scope fields do not match the original issuance , will fail Step 3 of the verification pipeline (root signature check). The security of this step relies on the unforgeability of Ed25519 signatures and the collision resistance of SHA-512 (used internally by Ed25519). An attacker who does not possess the issuer’s private key cannot produce a valid root signature for a modified token.

10.3. Chain Tampering

Modification of any hop in chain , including removal, reordering, or field modification , will cause hop signature verification (Step 5) to fail for the tampered hop and all subsequent hops, because each hop signature covers all previous hops. Insertion of a fabricated hop will similarly fail unless the attacker possesses the issuer’s private key.

10.4. Replay Attack Defense

HDP provides two orthogonal replay defenses:

1. *Expiry.* Tokens are short-lived (expires_at, 24h default). An expired token is rejected at Step 2 regardless of network conditions.

2. *Session binding.* The token carries the session_id established out-of-band between issuer and verifier. A token is valid only within the session for which it was issued. Even a non-expired token cannot be replayed across sessions.

Together, these defenses ensure that a stolen token is useful to an attacker only within the original session and only before it expires. Applications with high security requirements SHOULD use short token lifetimes (minutes, not hours).

10.5. Prompt Injection

Prompt injection attacks attempt to cause an agent to act as if it received instructions from a legitimate principal, when in fact the instructions originate from adversarial content in the agent's environment (e.g., a malicious web page or document). HDP mitigates but does not fully prevent this attack.

An HDP-aware agent SHOULD refuse to extend a token's chain with an action_summary that contradicts the token's scope.intent. However, the protocol cannot enforce this semantically , the comparison between action intent and token scope is application-defined.

The primary mitigation HDP provides is that injected actions MUST be recorded in the chain to be valid, creating an auditable record that the action occurred. Post-hoc detection of prompt injection attacks is thereby supported.

10.6. Key Management

The security of all HDP guarantees depends on the confidentiality of the issuer's Ed25519 private key. Implementations MUST:

* Store private keys in a secrets manager, HSM, or equivalent secure enclave. Private keys MUST NOT be stored in source code, configuration files, or environment variables in production.

* Use distinct key pairs per environment (development, staging, production).

* Support key rotation by issuing new tokens with a new kid while maintaining the old key in the verifier's registry until all tokens signed with it have expired.

10.7. Offline Verification Guarantee

HDP makes a strong architectural guarantee: a correct implementation of the 7-step verification pipeline requires no network calls, no registry lookups, and no third-party contact. The complete trust state required for verification is:

* The issuer's Ed25519 public key (32 bytes).

* The current session identifier (string).

* The current time (for expiry checking).

This guarantee enables HDP verification in air-gapped environments, edge deployments with intermittent connectivity, and latency-sensitive contexts where a network round-trip before every action is unacceptable.

11. IANA Considerations

11.1. HTTP Header Field Registration

This document requests registration of the following HTTP header fields in the "Hypertext Transfer Protocol (HTTP) Field Name Registry" maintained at <https://www.iana.org/assignments/http-fields/>.

Header Field Name: X-HDP-Token

Status: provisional

Reference: This document, Section 8.1

Comments: Carries a base64url-encoded HDP token for agentic delegation provenance.

Header Field Name: X-HDP-Token-Ref

Status: provisional

Reference: This document, Section 8.2

Comments: Carries the UUID token_id of an HDP token stored by reference.

11.2. Media Type Registration

This document requests registration of the media type application/hdp-token+json in the "Media Types" registry for use in contexts where the Content-Type of an HDP token must be explicitly identified.

12. Comparison with Related Work

12.1. IPP (draft-haberkamp-ipp-00)

The Intent Provenance Protocol [I-D.haberkamp-ipp] and HDP address the same root problem with different architectural trade-offs. The key differences are:

1. *Revocation model.* IPP requires agents to poll a central revocation registry at a configurable endpoint before every action, with a recommended interval of 5,000 milliseconds. If the registry is unreachable, agents cannot safely act. HDP uses short-lived tokens with session_id binding as the revocation mechanism; no registry polling is required at any point.

2. *Trust anchor.* IPP tokens contain a genesis_seal , a cryptographic artifact linking every token to the specification author's public key at https://ipp.khsovereign.com/keys/founding_public.pem. Self-hosted IPP deployments are cryptographically bound to this third-party key. HDP tokens carry no genesis seal and no spec-level attribution; any organization can issue and verify HDP tokens without anchoring to a third party.

3. *Identity model.* IPP mandates W3C DID Core-conformant principal identifiers. HDP supports id_type: "opaque" as a first-class option, making DID infrastructure optional rather than required.

These are design choices, not defects. Deployments with reliable connectivity to a central registry, existing DID infrastructure, and a requirement for mid-chain revocation may prefer IPP. Deployments that prioritize offline operability, self-sovereignty, and minimal infrastructure may prefer HDP.

12.2. OAuth 2.0 Token Exchange (RFC 8693)

OAuth 2.0 Token Exchange [RFC8693] defines a mechanism for exchanging one security token for another, including delegation and impersonation use cases. HDP and RFC 8693 are complementary rather than competing: RFC 8693 governs access token issuance and delegation in an OAuth 2.0 authorization server context, while HDP governs the provenance record that travels with an agentic task regardless of the authentication mechanism used.

HDP tokens do not replace OAuth access tokens. An agent framework MAY use OAuth 2.0 for resource authorization and HDP for delegation provenance simultaneously.

12.3. JSON Web Token (RFC 7519)

JSON Web Token [RFC7519] provides a general-purpose signed claims format. HDP differs from JWT in three respects:

* HDP tokens carry an append-only, multi-party-signed delegation chain (chain) that has no equivalent in the JWT standard claims set.

* HDP uses RFC 8785 canonical JSON for signing payloads, rather than the base64url-encoded header.payload convention used by JWS (RFC 7515). This allows direct JSON manipulation without base64 decoding.

* HDP's verification pipeline is domain-specific to agentic delegation (session binding, hop verification, max_hops) rather than general-purpose.

12.4. UCAN (User Controlled Authorization Networks)

UCAN defines a capability-based authorization token system using JWT with chained delegation. HDP and UCAN share the concept of delegation chains but differ significantly in scope: UCAN is a general capability authorization system, while HDP is specifically a provenance record for human-authorized agentic tasks. HDP makes no claims about capability enforcement; UCAN tokens carry executable capabilities that are enforced by receiving systems.

13. Normative References

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

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

[RFC8032] Josefsson, S. and I. Liusvaara, "Edwards-Curve Digital Signature Algorithm (EdDSA)", RFC 8032, DOI 10.17487/RFC8032, January 2017, <https://www.rfc-editor.org/rfc/rfc8032>.

[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/rfc/rfc8785>.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <https://www.rfc-editor.org/rfc/rfc4648>.

14. Informative References

[I-D.haberkamp-ipp]
Haberkamp, M., "Intent Provenance Protocol (IPP)", Work in Progress, Internet-Draft, draft-haberkamp-ipp-00, 2024, <https://datatracker.ietf.org/doc/html/draft-haberkamp-ipp-00>.

[RFC8693] Jones, M., Nadalin, A., Campbell, B., Bradley, J., and C. Liu, "OAuth 2.0 Token Exchange", RFC 8693, DOI 10.17487/RFC8693, January 2020, <https://www.rfc-editor.org/rfc/rfc8693>.

[RFC7519] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", RFC 7519, DOI 10.17487/RFC7519, May 2015, <https://www.rfc-editor.org/rfc/rfc7519>.

[W3C.DID] Sporny, M., Longley, D., Sabadello, M., Reed, D., Steele, O., and C. Allen, "Decentralized Identifiers (DIDs) v1.0", W3C Recommendation did-core, July 2022, <https://www.w3.org/TR/did-core/>.

[HDP-SPEC] Helixar Limited, "Human Delegation Provenance Protocol v0.1 Specification", 2026, <https://helixar.ai/labs/hdp>.

[HDP-IMPL] Helixar Limited, "HDP TypeScript Reference Implementation", 2026, <https://github.com/Helixar-AI/HDP>.

Appendix A. Complete Token Example

The following is a complete HDP token with a two-hop delegation chain, for illustrative purposes. Signature values are truncated.

```
{
  "hdp": "0.1",
  "header": {
    "token_id"   : "550e8400-e29b-41d4-a716-446655440000",
    "issued_at"  : 1711483200000,
    "expires_at" : 1711569600000,
    "session_id" : "sess-20260326-abc123",
    "version"    : "0.1"
  },
  "principal": {
    "id"           : "usr_alice_opaque",
    "id_type"      : "opaque",
    "display_name" : "Alice Chen"
  },
  "scope": {
    "intent"              : "Analyze Q1 sales data and produce a report.",
```

```
      "authorized_tools"    : ["database_read", "file_write"],
      "data_classification" : "confidential",
      "network_egress"      : false,
      "persistence"         : true,
      "max_hops"            : 3
    },
    "chain": [
      {
        "seq"            : 1,
        "agent_id"       : "orchestrator-v2",
        "agent_type"     : "orchestrator",
        "timestamp"      : 1711483260000,
        "action_summary" : "Decompose analysis task; delegate to sub-agents.",
        "parent_hop"     : 0,
        "hop_signature"  : "base64url-sig-1..."
      },
      {
        "seq"            : 2,
        "agent_id"       : "sql-agent-v1",
        "agent_type"     : "sub-agent",
        "timestamp"      : 1711483320000,
        "action_summary" : "Execute read query against sales database.",
        "parent_hop"     : 1,
        "hop_signature"  : "base64url-sig-2..."
      }
    ],
    "signature": {
      "kid"   : "alice-signing-key-v1",
      "alg"   : "Ed25519",
      "value" : "base64url-root-sig..."
    }
  }
```

Change Log

This section will be removed before publication as an RFC.

draft-helixar-hdp-agentic-delegation-00: Initial submission.
Specifies HDP v0.1 token structure, signing, verification pipeline, re-authorization, multi-principal delegation, transport, privacy considerations, and security analysis.

Author’s Address

Helixar Limited
Helixar Limited
Email: protocol@helixar.ai
URI: https://helixar.ai