Network Working Group
Internet-Draft
Intended status: Informational
Expires: 21 November 2026

T. Baur
Baur Software

20 May 2026

Principal Agent Protocol (PAP)
draft-baur-pap-00

Abstract

This document specifies the Principal Agent Protocol (PAP), a cryptographic protocol for human-controlled agent-to-agent transactions. PAP establishes a trust model rooted in human principals, defines hierarchical delegation through signed mandates, enforces context minimization through selective disclosure at the protocol level, and provides session ephemerality as a structural guarantee. The protocol uses no novel cryptographic primitives and requires no central registry, token economy, or trusted third party.

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 21 November 2026.

Copyright Notice

Table of Contents

1. Introduction

1.1. Problem Statement

Existing agent-to-agent protocols authenticate agents as platform entities, not as delegates of human principals. None enforce context minimization at the protocol level. Disclosure is implementation-dependent. Session ephemerality is undefined. Execution isolation is absent—agents run in the same address space as the orchestrator or other services, creating blast radius problems even when disclosure is minimized. Economic models underneath these protocols are compatible with platform capture through cloud compute metering.

1.2. Design Goals

PAP is designed to satisfy the following goals:

1. The human principal is the root of trust for every transaction.

2. Context disclosure is enforced by the protocol at the request boundary (via SD-JWT).
3. Execution is isolated at the process boundary via OS-level capabilities.
4. Sessions are ephemeral by design; no persistent correlation.
5. Delegation is hierarchical with cryptographically enforced bounds.
6. Co-signed receipts prove both disclosure scope and execution constraints.
7. No novel cryptography, no token economy, no central registry.
8. Any compliant implementation MUST be buildable from this document alone, without reference to a specific programming language.

1.3. Protocol Overview

A PAP transaction involves:

* A *human principal* who holds a device-bound keypair.
* An *orchestrator agent* operating under a root mandate.
* One or more *downstream agents* operating under delegated mandates, each executing in sandboxed isolation.
* A *marketplace* for agent discovery and disclosure filtering.
* A *6-phase session handshake* between pairs of agents.
* *Request boundary security* via SD-JWT selective disclosure (minimize what the agent sees).
* *Execution boundary security* via OS sandboxing (minimize what the agent can do).
* *Co-signed receipts* recording property references and enforcement proof, never values.

2. Conventions and Terminology

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 BCP 14 [RFC 2119] [RFC 8174] when, and only when, they appear in all capitals, as shown here.

2.1. Definitions

Principal: A human user who holds the root keypair and is the ultimate authority over all agent actions taken on their behalf.

Orchestrator: An agent that holds the root mandate from the principal. The orchestrator is the only agent that MAY hold the principal’s full context. It delegates scoped mandates to downstream agents.

Mandate: A signed authorization object that specifies what an agent is permitted to do, what context it may disclose, and when the authorization expires.

Mandate Chain: An ordered sequence of mandates from root to leaf, each cryptographically linked to its parent.

Scope: The set of actions a mandate permits. Deny-by-default: an empty scope permits nothing.

Disclosure Set: The set of context classes an agent holds and the conditions under which they may be shared.

Capability Token: A single-use, signed authorization to open a session with a specific agent for a specific action.

Session DID: An ephemeral did:key identifier generated for a single session and discarded at session close.

Receipt: A co-signed record of a transaction that contains property type references but never property values.

Decay State: The lifecycle state of a mandate as it approaches or passes its TTL without renewal.

3. Trust Model and Threat Model

3.1. Trust Hierarchy

The PAP trust hierarchy is:

```
Human Principal (device-bound keypair, root of trust)
  +-- Orchestrator Agent (root mandate, full principal context)
       +-- Downstream Agent (task mandate, scoped context)
            +-- Marketplace Agent (own principal chain)
```

The principal's device-bound keypair is the sole root of trust. Every agent in a transaction MUST carry a cryptographically verifiable mandate chain traceable to this root.

3.2. Trust Assumptions

Assumption	Verification Method
Principal keypair not compromised	WebAuthn device binding (Section 4.3)

Orchestrator delegates correctly	Mandate chain verification (Section 5.6)
Session keys not leaked	Single-use per session, discarded at close
Clocks approximately synchronized	RFC 3339 timestamps; receivers SHOULD reject tokens with skew exceeding implementation-defined thresholds
Ed25519 not broken	Cryptographic library security; algorithm agility reserved for future versions

Table 1

3.3. Threat Model

PAP is designed to defend against the following threats:

T1. Context profiling. An adversary correlates a principal's transactions across sessions to build a behavioral profile. _Mitigation:_ Ephemeral session DIDs (Section 6.3) ensure each session is cryptographically unlinkable.

T2. Over-disclosure. An agent discloses more principal context than the principal authorized. _Mitigation:_ SD-JWT selective disclosure (Section 7) structurally prevents disclosure of claims not included in the disclosure set. Marketplace filtering (Section 9.3) excludes agents whose requirements exceed the mandate before any session is established.

T3. Delegation bypass. A downstream agent acts outside its delegated scope. _Mitigation:_ Scope containment (Section 5.4) and TTL bounds (Section 5.5) are verified cryptographically at each level of the mandate chain.

T4. Replay attacks. An adversary replays a captured capability token to open an unauthorized session. _Mitigation:_ Nonce consumption (Section 6.2) ensures each token is single-use.

T5. Mandate tampering. An adversary modifies a mandate in the chain. _Mitigation:_ Parent hash binding (Section 5.3) and Ed25519 signatures (Section 5.2) detect any modification.

T6. Platform capture. A platform operator accumulates control over agent transactions through infrastructure dependency. _Mitigation:_ Federated discovery (Section 10), no central registry, no token economy, principal-held keys. Marketplace registries MUST NOT rank query results by operator metrics (Section 9.6) — ranking power is platform capture power. Trust evaluation is the principal's responsibility.

T7. Payment linkability. A payment is correlated with the principal's identity. _Mitigation:_ Chaumian ecash blind-signed tokens (Section 13.1) provide unlinkable proof of value transfer.

3.4. Explicit Non-Goals

The following are explicitly out of scope for PAP:

1. Compatibility with token economy monetization.
2. Enclave-as-equivalent-to-local trust models.
3. Identity recovery through platform operators.
4. Central registries for agent discovery.
5. Runtime scope expansion of mandates.
6. Arbitrary code execution in the orchestrator context.
7. Any extension that trades trust guarantees for adoption ease.

4. Identity Layer

4.1. DID Method

PAP uses the did:key method as defined in [DID-KEY]. All identifiers MUST use Ed25519 public keys with the following derivation:

did:key:z<base58btc(0xed01 || public_key_bytes)>

Where: - 0xed01 is the multicodec prefix for Ed25519 public keys. - public_key_bytes is the 32-byte Ed25519 public key. - base58btc is Bitcoin's base58 encoding. - The z prefix indicates base58btc multibase encoding.

Implementations MUST support did:key resolution by extracting the public key bytes from the DID string:

1. Strip the did:key:z prefix.
2. Base58-decode the remainder.
3. Verify the first two bytes are 0xed and 0x01.
4. The remaining 32 bytes are the Ed25519 public key.

4.2. DID Document

A DID document for a PAP identity MUST conform to [DID-CORE] and contain:

```
{
  "@context": "https://www.w3.org/ns/did/v1",
  "id": "did:key:z...",
  "verificationMethod": [{
    "id": "did:key:z...#key-1",
    "type": "Ed25519VerificationKey2020",
    "controller": "did:key:z...",
    "publicKeyMultibase": "z<base58btc(0xed01 ++ public_key_bytes)>"
  }],
  "authentication": ["did:key:z...#key-1"]
}
```

A DID document MUST NOT contain any personal information. It contains only the public key and verification method reference.

4.3. Principal Keypair

The principal keypair is the root of trust. It MUST be an Ed25519 keypair. In production deployments, the private key SHOULD be bound to a hardware authenticator via WebAuthn [WEBAUTHN].

Implementations MUST support the PrincipalSigner interface:

* did() -> String -- The did:key identifier.
* sign(message: bytes) -> bytes -- Ed25519 signature (64 bytes).
* verifying_key() -> Ed25519PublicKey -- The public key.

Implementations MAY use software keys for development and testing. Production deployments SHOULD use WebAuthn-backed keys.

4.4. Session Keypair

A session keypair is an ephemeral Ed25519 keypair generated fresh for each protocol session. Session keypairs:

* MUST be generated using a cryptographically secure random number generator.
* MUST NOT be derived from or linked to the principal keypair.
* MUST be discarded when the session closes.
* MUST NOT be persisted to stable storage.

The session DID is derived using the same did:key method as the principal DID. An observer MUST NOT be able to determine whether a did:key identifier represents a principal or a session key.

5. Mandate Structure and Delegation Rules

5.1. Mandate Object

A mandate is the core delegation primitive. It authorizes an agent to perform specific actions with specific context. A mandate MUST contain the following fields:

Field	Type	Required	Description
principal_did	String	REQUIRED	DID of the human principal (root of trust)
agent_did	String	REQUIRED	DID of the agent receiving this mandate
issuer_did	String	REQUIRED	DID of the entity signing this mandate
parent_mandate_hash	String or null	REQUIRED	SHA-256 hash of the parent mandate, or null for root mandates
scope	Scope	REQUIRED	Permitted actions (Section 5.4)
disclosure_set	DisclosureSet	REQUIRED	Context classes and sharing conditions (Section 5.4.3)
ttl	DateTime	REQUIRED	Expiry timestamp (RFC 3339)
decay_state	DecayState	REQUIRED	Current lifecycle state (Section 5.7)
issued_at	DateTime	REQUIRED	Issuance timestamp (RFC 3339)
payment_proof	PaymentProof or null	OPTIONAL	ZK payment commitment (Section 13.1)
signature	String or null	OPTIONAL	Ed25519 signature (base64url-no-pad)

Table 2

5.2. Mandate Signing

A mandate MUST be signed by the issuer's Ed25519 signing key.

The canonical form for signing MUST be computed as follows:

1. Construct a JSON object containing all mandate fields EXCEPT signature.
2. DateTime fields MUST be serialized as RFC 3339 strings.
3. Null fields MUST be included as JSON null.
4. Serialize the JSON object to bytes.
5. Compute the Ed25519 signature over these bytes.
6. Encode the 64-byte signature using base64url without padding (RFC 4648 Section 5, no = padding).

The canonical JSON object MUST contain exactly these keys:

```
{
  "principal_did": "...",
  "agent_did": "...",
  "issuer_did": "...",
  "parent_mandate_hash": null,
  "scope": { ... },
  "disclosure_set": { ... },
  "ttl": "2026-03-15T20:00:00+00:00",
  "issued_at": "2026-03-15T16:00:00+00:00",
  "payment_proof": null
}
```

5.3. Mandate Hashing

The mandate hash is used for parent-child linking in delegation chains. It MUST be computed as:

1. Compute the canonical form (Section 5.2, step 1-4).
2. Apply SHA-256 to the canonical bytes.
3. Encode the 32-byte digest using base64url without padding.

The hash MUST be deterministic: the same mandate MUST always produce the same hash.

5.4. Scope

5.4.1. 5.4.1. Scope Object

A scope defines the set of permitted actions. It is deny-by-default: an agent with an empty scope MUST NOT perform any action.

```
{
  "actions": [
    {
      "action": "schema:SearchAction",
      "object": "schema:WebPage",
      "conditions": {}
    }
  ]
}
```

Field	Type	Required	Description
actions	Array of ScopeAction	REQUIRED	The permitted actions

Table 3

5.4.2. 5.4.2. ScopeAction Object

Field	Type	Required	Description
action	String	REQUIRED	Schema.org action type (e.g., schema:SearchAction)
object	String or null	OPTIONAL	Schema.org object type constraint (e.g., schema:Flight)
conditions	Object	OPTIONAL	Protocol-level conditions (key-value pairs). Default: empty object.

Table 4

Action and object type references MUST use the schema: prefix for Schema.org vocabulary. Implementations MAY define additional namespaced prefixes for domain-specific vocabularies.

5.4.3. 5.4.3. DisclosureSet Object

The disclosure set defines what context an agent holds and the conditions for sharing it.

```
{
  "entries": [
    {
      "type": "schema:Person",
      "permitted_properties": ["schema:name", "schema:nationality"],
      "prohibited_properties": ["schema:email", "schema:telephone"],
      "session_only": true,
      "no_retention": true
    }
  ]
}
```

Field	Type	Required	Description
entries	Array of DisclosureEntry	REQUIRED	The disclosure entries

Table 5

5.4.4. 5.4.4. DisclosureEntry Object

Field	Type	Required	Description
type	String	REQUIRED	Schema.org type (e.g., schema:Person)
permitted_properties	Array of String	REQUIRED	Properties the agent MAY disclose
prohibited_properties	Array of String	REQUIRED	Properties the agent MUST NOT disclose
session_only	Boolean	OPTIONAL	If true, disclosed data is valid only for the session duration. Default: false.
no_retention	Boolean	OPTIONAL	If true, the receiving party MUST NOT retain disclosed data

				beyond the session. Default: false.

Table 6

Property references MUST use Schema.org property names with the schema: prefix.

Property Reference Format: When used in receipts or marketplace advertisements, a fully qualified property reference is formed as {type}.{property}, e.g., schema:Person.schema:name.

5.4.5. 5.4.4.1. TEE Requirement for No-Retention Disclosures

When a disclosure entry has no_retention set to true, the receiving agent MUST provide TEE attestation (Section 13.6) during session establishment. If the receiving agent cannot provide valid TEE attestation, the initiating agent MUST NOT disclose properties from that entry.

Without TEE attestation, no_retention is a contractual constraint only — the protocol cannot enforce data deletion on an untrusted host. Implementations SHOULD clearly communicate this limitation to principals when TEE attestation is unavailable.

An implementation's disclosure validation MUST return one of three states to the caller:

State	Meaning
NotRequired	No no_retention entries in the disclosure set
TeeEnforced	TEE attestation present; retention constraint is cryptographic
ContractualOnly	No TEE available; no_retention is a contractual term only

Table 7

Implementations that support the TEE extension (Section 13.6) MUST treat ContractualOnly as an error. Implementations without TEE support MAY proceed with ContractualOnly but MUST expose this state to the caller so the principal can make an informed decision.

5.4.6. 5.4.5. Scope Containment

A child scope S_c is *contained by* a parent scope S_p (written S_c <= S_p) if and only if for every action A_c in S_c, there exists an action A_p in S_p such that:

1. A_c.action == A_p.action
2. If A_p.object is non-null, then A_c.object MUST equal A_p.object.
3. If A_p.object is null, then A_c.object MAY be any value (including null).
4. If A_p.object is non-null and A_c.object is null, the containment check MUST fail. A child MUST NOT broaden an object constraint.

5.5. Delegation Rules

When an agent delegates a mandate to a child agent, the following rules MUST be enforced:

R1. Scope Containment: The child mandate's scope MUST be contained by the parent mandate's scope (Section 5.4.5). If scope containment fails, the delegation MUST be rejected.

R2. TTL Bound: The child mandate's ttl MUST NOT exceed the parent mandate's ttl. If the child TTL exceeds the parent TTL, the delegation MUST be rejected.

R3. Parent Hash Binding: The child mandate's parent_mandate_hash MUST equal the hash (Section 5.3) of the parent mandate's canonical form.

R4. Issuer Chain: The child mandate's issuer_did MUST equal the parent mandate's agent_did. The child mandate MUST be signed by the parent mandate's agent_did key.

R5. Principal Propagation: The child mandate's principal_did MUST equal the parent mandate's principal_did.

R6. Root Mandate: A root mandate MUST have parent_mandate_hash set to null. A root mandate's issuer_did MUST equal its principal_did.

5.6. Mandate Chain Verification

A mandate chain is an ordered array of mandates [M_0, M_1, ..., M_n] where M_0 is the root mandate. Verification MUST proceed as follows:

1. M_0.parent_mandate_hash MUST be null.
2. M_0.signature MUST verify against the principal's public key.

3. For each i from 1 to n: a. M_i.parent_mandate_hash MUST equal hash(M_{i-1}). b. M_i.scope MUST satisfy scope containment against M_{i-1}.scope (Section 5.4.5). c. M_i.ttl MUST NOT exceed M_{i-1}.ttl. d. M_i.signature MUST verify against the public key of M_{i-1}.agent_did.

If any check fails, the entire chain MUST be rejected.

5.7. Decay State Machine

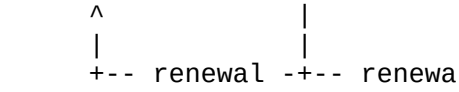
A mandate’s decay state tracks its lifecycle as the TTL progresses. The decay state MUST be one of:

State	Description
Active	Full scope, within TTL
Degraded	Reduced scope, TTL within decay window, renewal pending
ReadOnly	No execution permitted, observation only, TTL expired
Suspended	No activity, awaiting principal review

Table 8

5.7.1. 5.7.1. State Transitions

The following transitions are valid:



From	To	Condition
Active	Degraded	Remaining TTL <= implementation-defined decay window
Degraded	ReadOnly	TTL expired without renewal
ReadOnly	Suspended	Implementation-defined timeout without principal action
Degraded	Active	Mandate renewed by issuer
ReadOnly	Active	Mandate renewed by issuer
Suspended	(none)	Suspended mandates MUST NOT be renewed. Principal MUST issue a new mandate.

Table 9

Any transition not listed above MUST be rejected.

5.7.2. 5.7.2. Decay Computation

An implementation SHOULD compute the current decay state as:

```
function compute_decay_state(mandate, decay_window_seconds):
  now = current_utc_time()
  if now > mandate.ttl:
    if mandate.decay_state == Suspended:
      return Suspended
    else:
      return ReadOnly
  else:
    remaining = mandate.ttl - now (in seconds)
    if remaining <= decay_window_seconds:
      return Degraded
    else:
      return Active
```

The decay_window_seconds parameter is implementation-defined. Implementations SHOULD document their chosen value.

6. Session Lifecycle

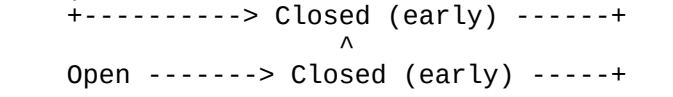
6.1. Session State Machine

A session tracks the state of a transaction between two agents. The session state MUST be one of:

State	Description
Initiated	Capability token presented, awaiting verification
Open	Handshake complete, session DIDs exchanged
Executed	Transaction executed within session
Closed	Session closed, ephemeral keys discarded

Table 10

Valid transitions:



From	To	Trigger
Initiated	Open	Session DID exchange completed
Initiated	Closed	Early termination (rejection or error)
Open	Executed	Action executed
Open	Closed	Early termination
Executed	Closed	Session close message sent

Table 11

Any transition not listed above MUST be rejected.

6.2. Capability Token

A capability token is a single-use authorization to open a session. It MUST contain the following fields:

Field	Type	Required	Description
id	String	REQUIRED	Unique token identifier (UUID v4)
target_did	String	REQUIRED	DID of the agent this token authorizes a session with
action	String	REQUIRED	Schema.org action type this token authorizes
nonce	String	REQUIRED	Single-use nonce (UUID v4), consumed on session initiation
issuer_did	String	REQUIRED	DID of the issuing agent (typically the orchestrator)
issued_at	DateTime	REQUIRED	Issuance timestamp (RFC 3339)
expires_at	DateTime	REQUIRED	Expiry timestamp (RFC 3339)
signature	String or null	OPTIONAL	Ed25519 signature (base64url-no-pad)

Table 12

6.2.1. 6.2.1. Token Signing

The token canonical form MUST be:

```
{
  "id": "...",
  "target_did": "...",
  "action": "...",
  "nonce": "...",
  "issuer_did": "...",
  "issued_at": "...",
  "expires_at": "..."
}
```

Signing follows the same procedure as mandate signing (Section 5.2).

6.2.2. 6.2.2. Token Verification

A receiving agent MUST verify a capability token as follows:

1. token.target_did MUST match the receiver’s DID.
2. token.nonce MUST NOT appear in the receiver’s consumed nonce set.
3. The current time MUST NOT exceed token.expires_at.
4. token.signature MUST verify against the public key of token.issuer_did.

If all checks pass, the receiver MUST immediately add token.nonce to its consumed nonce set. A nonce, once consumed, MUST never be accepted again.

6.3. Six-Phase Handshake

The session handshake consists of six phases. Each phase involves a message exchange between the initiating agent (I) and the receiving agent (R).

Phase	Direction	Message	Data
1a	I -> R	TokenPresentation	CapabilityToken
1b	R -> I	TokenAccepted	session_id, receiver_session_did
	R -> I	TokenRejected	reason (terminates handshake)
2a	I -> R	SessionDidExchange	initiator_session_did
2b	R -> I	SessionDidAck	(empty)
3a	I -> R	DisclosureOffer	disclosures (may be empty array)
3b	R -> I	DisclosureAccepted	(empty)
4	R -> I	ExecutionResult	result (Schema.org JSON-LD)
5a	I -> R	ReceiptForCoSign	half-signed TransactionReceipt
5b	R -> I	ReceiptCoSigned	fully co-signed TransactionReceipt
6a	I -> R	SessionClose	session_id
6b	R -> I	SessionClosed	(empty)

6.3.1. 6.3.1. Phase 1: Token Presentation

The initiating agent presents a signed capability token. The receiving agent verifies the token (Section 6.2.2).

On acceptance, the receiver MUST: 1. Generate a fresh session keypair (Section 4.4). 2. Create a session in the Initiated state. 3. Return a TokenAccepted message containing the session ID and the receiver’s ephemeral session DID.

On rejection, the receiver MUST return a TokenRejected message with a reason string. The handshake terminates.

6.3.2. 6.3.2. Phase 2: Ephemeral DID Exchange

The initiating agent generates its own fresh session keypair and sends a SessionDidExchange message containing its session DID.

On receipt, the receiver MUST: 1. Transition the session state from Initiated to Open. 2. Store the initiator’s session DID. 3. Return a SessionDidAck message.

After Phase 2, both parties have exchanged ephemeral session DIDs. All subsequent envelope signatures (Section 8.2) MUST use session keys.

6.3.3. 6.3.3. Phase 3: Disclosure

The initiating agent sends a DisclosureOffer containing an array of SD-JWT disclosures (Section 7). The array MAY be empty for zero-disclosure sessions.

The receiver MUST: 1. Verify each disclosure against the SD-JWT commitment (Section 7.3). 2. Return a DisclosureAccepted message.

If disclosure verification fails, the receiver SHOULD return an Error message and close the session.

6.3.4. 6.3.4. Phase 4: Execution

The receiver executes the requested action and returns an ExecutionResult message containing a Schema.org JSON-LD result object.

The session state MUST transition from Open to Executed.

6.3.5. 6.3.5. Phase 5: Receipt Co-Signing

The initiating agent constructs a TransactionReceipt (Section 11), signs it with its session key, and sends it as ReceiptForCoSign.

The receiving agent MUST: 1. Verify the initiator’s signature on the receipt. 2. Add its own co-signature using its session key. 3. Return the fully co-signed receipt as ReceiptCoSigned.

6.3.6. 6.3.6. Phase 6: Session Close

Either party MAY initiate session close by sending a SessionClose message containing the session ID.

On receipt of SessionClose, the other party MUST: 1. Return a SessionClosed message. 2. Transition the session state to Closed. 3. Discard all ephemeral session keys.

After Phase 6, both parties MUST discard their session keypairs. Session DIDs MUST NOT be reused.

7. SD-JWT Disclosure Protocol

7.1. Overview

PAP uses Selective Disclosure JWT (SD-JWT) as defined in [SD-JWT-08] for context disclosure during the session handshake. SD-JWT allows the principal to hold multiple claims but disclose only those permitted by the mandate.

7.2. SD-JWT Object

An SD-JWT MUST contain:

Field	Type	Required	Description
issuer	String	REQUIRED	DID of the claim issuer (typically the principal)
claims	Object	REQUIRED (private)	All claims as key-value pairs
salts	Object	REQUIRED (private)	Per-claim random salts (UUID v4)
signature	String or null	OPTIONAL	Ed25519 signature over commitment bytes (base64url-no-pad)

Table 13

The claims and salts fields are private to the holder and MUST NOT be transmitted in their entirety. Only selected disclosures (Section 7.3) are transmitted.

7.3. Disclosure Object

A disclosure reveals a single claim. It MUST contain:

Field	Type	Required	Description
salt	String	REQUIRED	The claim-specific random salt
key	String	REQUIRED	The claim key
value	Any JSON value	REQUIRED	The claim value

Table 14

7.4. Commitment Computation

The SD-JWT commitment is signed to bind all possible disclosures.

1. For each claim (key, value) with salt s:

    * Construct: {"salt": s, "key": key, "value": value}
    * Hash: SHA-256(JSON_bytes(disclosure))
    * Encode: base64url-no-pad

2. Collect all hashes and sort lexicographically.

3. Construct commitment bytes:

```
{
 "issuer": "<issuer_did>",
 "disclosure_hashes": ["<sorted_hash_1>", "<sorted_hash_2>", ...]
}
```

4. Sign: Ed25519_sign(JSON_bytes(commitment))

7.5. Disclosure Verification

A verifier MUST:

1. Verify the SD-JWT signature over the commitment bytes using the issuer’s public key.
2. For each received disclosure: a. Compute hash = base64url(SHA-256(JSON_bytes(disclosure))). b. Verify that hash is present in the signed disclosure_hashes array.

If any disclosure hash is not found in the commitment, the verification MUST fail.

7.6. Zero-Disclosure Sessions

A session MAY proceed with zero disclosures. In this case:

* The DisclosureOffer message carries an empty disclosures array.
* The SD-JWT signature MUST still verify (the commitment contains hashes for all claims, but none are revealed).
* The receiver MUST accept an empty disclosure set without error.

8. Protocol Messages and Envelope

8.1. Protocol Message Types

All protocol messages are serialized as JSON objects with a type discriminator field. The following message types are defined:

Type	Phase	Direction	Fields
TokenPresentation	1	I->R	token: CapabilityToken
TokenAccepted	1	R->I	session_id: String, receiver_session_did: String
TokenRejected	1	R->I	reason: String
SessionDidExchange	2	I->R	initiator_session_did: String
SessionDidAck	2	R->I	(no fields)
DisclosureOffer	3	I->R	disclosures: Array of JSON values
DisclosureAccepted	3	R->I	(no fields)
ExecutionResult	4	R->I	result: JSON value (Schema.org JSON-LD)
ReceiptForCoSign	5	I->R	receipt: TransactionReceipt
ReceiptCoSigned	5	R->I	receipt: TransactionReceipt
SessionClose	6	Either	session_id: String
SessionClosed	6	Either	(no fields)
Error	Any	Either	code: String, message: String

Table 15

8.2. Envelope

Protocol messages are transmitted inside an envelope that provides routing, sequencing, and integrity.

Field	Type	Required	Description
id	String	REQUIRED	Unique envelope identifier (UUID v4)
session_id	String	REQUIRED	Session this envelope belongs to
sender	String	REQUIRED	DID of the sender
recipient	String	REQUIRED	DID of the intended recipient
sequence	Integer	REQUIRED	Monotonically increasing sequence number within the session
payload	ProtocolMessage	REQUIRED	The protocol message
timestamp	DateTime	REQUIRED	ISO 8601 timestamp
signature	Bytes or null	OPTIONAL	Ed25519 signature over signable bytes

Table 16

8.2.1. 8.2.1. Envelope Signing

The signable bytes for an envelope MUST be computed as:

```
SHA-256(session_id_bytes || sequence_big_endian_8_bytes || payload_json_bytes)
```

Where || denotes concatenation and sequence_big_endian_8_bytes is the sequence number as an 8-byte big-endian integer.

Before Phase 2 (DID exchange), the signature field MAY be null because the capability token carries its own signature from the issuer.

After Phase 2, all envelopes MUST be signed by the sender’s ephemeral session key.

8.2.2. 8.2.2. Envelope Verification

The recipient MUST:

1. Verify recipient matches its own DID.
2. Verify sequence is strictly greater than the last received sequence number for this session.
3. If signature is present, verify it against the sender’s session public key.

9. Marketplace Advertisement Schema

9.1. Agent Advertisement

An agent advertisement declares an agent’s capabilities, disclosure requirements, and return types. Advertisements use Schema.org vocabulary and JSON-LD structure.

Field	Type	Required	Description
@context	String	REQUIRED	MUST be "https://schema.org"
@type	String	REQUIRED	MUST be "schema:Service"
name	String	REQUIRED	Human-readable agent name
provider	Provider	REQUIRED	Provider organization (Section 9.2)
capability	Array of String	REQUIRED	Schema.org action types the agent can perform
object_types	Array of String	REQUIRED	Schema.org object types the agent operates on
requires_disclosure	Array of String	REQUIRED	Fully qualified property references the agent requires (e.g., schema:Person.name)
returns	Array of String	REQUIRED	Schema.org types the agent returns
ttl_min	Integer	OPTIONAL	Minimum session TTL in seconds. Default: 300.
signed_by	String	REQUIRED	DID that signed this advertisement
signature	String or null	OPTIONAL	Ed25519 signature (base64url-no-pad)

Table 17

9.2. Provider Object

Field	Type	Required	Description
@type	String	REQUIRED	MUST be "schema:Organization"
name	String	REQUIRED	Organization name
did	String	REQUIRED	Operator DID

Table 18

9.3. Disclosure Filtering

A marketplace registry MUST support two query modes:

Query by action: Return all advertisements whose capability array contains the requested action type.

Query by action with disclosure satisfiability: Return only advertisements where: 1. The capability array contains the requested action type, AND 2. Every entry in requires_disclosure is present in the caller's available properties list.

This filtering MUST occur before any mandate is issued or session is established. Agents whose disclosure requirements exceed the principal's authorization MUST be excluded. The principal MUST NOT be asked to over-disclose.

9.4. Advertisement Signing

The canonical form for advertisement signing MUST include all fields except signature:

```
{
  "@context": "https://schema.org",
  "@type": "schema:Service",
  "name": "...",
  "provider": { ... },
  "capability": [...],
  "object_types": [...],
  "requires_disclosure": [...],
  "returns": [...],
  "ttl_min": 300,
  "signed_by": "did:key:z..."
}
```

Signing follows the same Ed25519/base64url-no-pad procedure as mandate signing (Section 5.2).

A marketplace registry MUST reject unsigned advertisements.

9.5. Advertisement Hashing

The content hash of an advertisement MUST be computed as:

base64url(SHA-256(canonical_bytes))

This hash is used for deduplication in federated registries (Section 10).

9.6. Operator Metrics

An agent advertisement MAY include an operator_metrics field containing self-reported operational statistics. Metrics are informational metadata for principal evaluation and MUST NOT be used by marketplace registries for ranking, sorting, or filtering query results.

Field	Type	Required	Description
total_receipts	Integer	OPTIONAL	Total co-signed transaction receipts
bilateral_attestations	Integer	OPTIONAL	Receipts with bilateral session attestation
unique_counterparties	Integer	OPTIONAL	Distinct counterparty session DIDs
action_types	Array of String	OPTIONAL	Distinct Schema.org action types performed
tee_sessions_pct	Number	OPTIONAL	Fraction of sessions with TEE attestation (0.0 to 1.0)
first_seen	DateTime	OPTIONAL	RFC 3339 timestamp of first registration
uptime_days	Integer	OPTIONAL	Days the operator has been active

Table 19

The operator_metrics field MUST be excluded from the advertisement content hash (Section 9.5) and signature computation (Section 9.4). Metrics change over time while the advertisement identity remains stable.

Anti-ranking requirement: Marketplace registries MUST return query results in insertion order. Registries MUST NOT rank, sort, or filter results based on operator metrics. The principal's orchestrator is responsible for evaluating metrics and making selection decisions. This requirement prevents marketplace registries from accumulating ranking power, which would constitute platform capture.

10. Federation Protocol

10.1. Overview

Federation enables independent marketplace registries to discover and share agent advertisements. Federation is peer-to-peer with no central coordinator.

10.2. Registry Peer

A federation peer is identified by:

Field	Type	Required	Description
did	String	REQUIRED	DID of the peer registry operator
endpoint	String	REQUIRED	HTTP(S) endpoint for federation API calls
last_sync	DateTime or null	OPTIONAL	Timestamp of last successful sync

Table 20

10.3. Federation Messages

Federation uses the following message types, discriminated by a type field:

Type	Direction	Fields	Description
QueryByAction	Request	action: String	Query for agents supporting an action
QueryResponse	Response	advertisements: Array of AgentAdvertisement	Matching advertisements
Announce	Request	advertisement: AgentAdvertisement	Announce a new local advertisement
AnnounceAck	Response	hash: String, accepted: Boolean	Acknowledge announcement
PeerList	Request	(none)	Request known peer list
PeerListResponse	Response	peers: Array of RegistryPeer	Known peers

Table 21

10.4. Federation Endpoints

A federation server MUST expose the following HTTP endpoints:

Method	Path	Request Body	Response Body
GET	/federation/query?action={action}	(none)	QueryResponse
POST	/federation/announce	Announce	AnnounceAck
GET	/federation/peers	(none)	PeerListResponse

Table 22

10.5. Content-Hash Deduplication

When merging remote advertisements, a federated registry MUST:

1. Compute the content hash of each advertisement (Section 9.5).
2. If the hash already exists in the local seen-hashes set, skip the advertisement.
3. If the advertisement has no signature, skip it.
4. Otherwise, register the advertisement and add its hash to the seen-hashes set.

This ensures idempotent synchronization and prevents duplicate entries.

10.6. Peer Discovery

A registry MAY discover new peers transitively:

1. Query a known peer's /federation/peers endpoint.
2. For each peer in the response not already known, add it to the local peer list.

Implementations SHOULD implement rate limiting and SHOULD validate that newly discovered peers are reachable before adding them.

10.7. Peer Trust Signals

A federation peer MAY present trust signals to establish credibility with other registries. Trust signals are additive — more signals increase confidence but no single signal is sufficient alone.

10.7.1. 10.7.1. Signal Categories

Signal	Weight	Description
Social vouching	Primary	Signed vouches from existing peers
TEE attestation	Supplementary	Hardware attestation of registry software
Operational history	Supplementary	Observable uptime and sync metrics
Domain verification	Supplementary	DNS or TLS proof of domain ownership

Table 23

A registry SHOULD require at least two signal categories before granting a peer full synchronization privileges.

10.7.2. 10.7.2. Peer Vouch

A peer vouch is a signed statement by an existing peer that they have evaluated the new peer and believe it operates a conformant registry.

Field	Type	Required	Description
voucher_did	String	REQUIRED	DID of the vouching peer
vouchee_did	String	REQUIRED	DID of the peer being vouched
timestamp	DateTime	REQUIRED	RFC 3339 timestamp
justification	String	REQUIRED	Structured reason for vouching
signature	String	REQUIRED	Ed25519 signature by voucher

Table 24

10.7.3. 10.7.3. Vouch Budget

To prevent vouch ring attacks (where colluding peers mutually vouch to create Sybil identities), implementations SHOULD enforce:

* *Vouch budget:* Each peer MAY issue at most 3 vouches per year.
* *Minimum age:* A peer MUST be registered for at least 90 days before it is eligible to vouch for others.
* *Probationary period:* Newly registered peers operate in probationary status for 60 days. During probation, a peer MAY receive advertisements but MUST NOT vouch for other peers.
* *Diverse trust paths:* The vouchers for a new peer SHOULD NOT all trace their own vouching chains through the same set of peers.

11. Receipt Format

11.1. Transaction Receipt

A transaction receipt is a co-signed record of a completed session. Receipts contain property type references only -- never values.

Field	Type	Required	Description
session_id	String	REQUIRED	Ephemeral session ID (not linked to principal)
action	String	REQUIRED	Schema.org action type executed
initiating_agent_did	String	REQUIRED	Ephemeral session DID of the initiator
receiving_agent_did	String	REQUIRED	Ephemeral session DID of the receiver
disclosed_by_initiator	Array of String	REQUIRED	Property references disclosed by the initiator
disclosed_by_receiver	Array of String	REQUIRED	Property references or operator statements from the receiver
executed	String	REQUIRED	Human-readable description of the action executed
returned	String	REQUIRED	Human-readable description of the result returned
timestamp	DateTime	REQUIRED	RFC 3339 timestamp
signatures	Array of String	REQUIRED	Co-signatures (base64url-no-pad)

Table 25

11.2. Receipt Signing

The canonical form for receipt signing MUST include all fields except signatures:

```
{
  "session_id": "...",
  "action": "...",
  "initiating_agent_did": "...",
  "receiving_agent_did": "...",
  "disclosed_by_initiator": [...],
  "disclosed_by_receiver": [...],
  "executed": "...",
  "returned": "...",
  "timestamp": "..."
}
```

11.3. Co-Signing Protocol

1. The initiator constructs a receipt from the completed session.
2. The initiator computes Ed25519_sign(canonical_bytes) using its session key and appends the base64url-no-pad encoded signature to signatures.
3. The initiator sends the half-signed receipt to the receiver.
4. The receiver verifies the initiator’s signature against the initiator’s session public key.
5. The receiver computes Ed25519_sign(canonical_bytes) using its session key and appends its signature to signatures.
6. The receiver returns the fully co-signed receipt.

11.4. Receipt Verification

To verify a co-signed receipt:

1. The signatures array MUST contain exactly 2 entries.
2. signatures[0] MUST verify against the initiator’s session public key.
3. signatures[1] MUST verify against the receiver’s session public key.

11.5. Privacy Properties

Receipts MUST NOT contain: - Personal data values (names, emails, etc.) - SD-JWT claim values - Raw execution inputs or outputs

Receipts MUST contain only: - Schema.org property type references (e.g., schema:Person.schema:name) - Operator-defined category references (e.g., operator:search_executed) - Human-readable action/ result descriptions

This ensures receipts are auditable by both principals without revealing the data exchanged in the transaction.

11.6. Session Attestation

A session attestation is a signed statement by a session participant recording their assessment of the session outcome.

Field	Type	Required	Description
session_id	String	REQUIRED	Session identifier
attester_did	String	REQUIRED	Ephemeral session DID of attester
outcome	String	REQUIRED	One of: fulfilled, partial, failed, disputed
action_type	String	REQUIRED	Schema.org action type executed
timestamp	DateTime	REQUIRED	RFC 3339 timestamp
signature	String	REQUIRED	Ed25519 signature by attester

Table 26

A receipt with attestations from both the initiating and receiving agents is *bilaterally attested*. Bilaterally attested receipts carry higher evidentiary weight for operator metric computation.

Attestations are per-action-type. An operator’s reputation in one action domain (e.g., schema:SearchAction) MUST NOT be conflated with reputation in another domain (e.g., schema:ReserveAction).

12. Verifiable Credential Envelope

12.1. Overview

PAP mandates MAY be wrapped in a W3C Verifiable Credential (VC) envelope for interoperability with existing credential ecosystems. The VC envelope is OPTIONAL; implementations MUST support bare mandates and MAY additionally support VC-wrapped mandates.

12.2. VC Structure

A PAP Verifiable Credential MUST conform to [VC-DATA-MODEL-2.0]:

```
{
  "@context": [
    "https://www.w3.org/ns/credentials/v2",
    "https://www.w3.org/ns/credentials/examples/v2"
  ],
  "id": "urn:uuid:<uuid-v4>",
  "type": ["VerifiableCredential", "PAPMandateCredential"],
  "issuer": "<issuer_did>",
  "issuanceDate": "<rfc3339>",
  "expirationDate": "<rfc3339>",
  "credentialSubject": { <mandate_payload> },
  "proof": {
    "type": "Ed25519Signature2020",
    "created": "<rfc3339>",
    "verificationMethod": "<did>#key-1",
    "proofPurpose": "assertionMethod",
    "proofValue": "<base64url-no-pad>"
  }
}
```

The type array MUST include both "VerifiableCredential" and "PAPMandateCredential" for discoverability.

12.3. Credential Signing

The canonical form for VC signing MUST include all fields except proof:

```
{
  "@context": [...],
  "id": "...",
  "type": [...],
  "issuer": "...",
  "issuanceDate": "...",
  "expirationDate": "..." or null,
  "credentialSubject": { ... }
}
```

The proofValue is base64url(Ed25519_sign(JSON_bytes(canonical))).

13. Extension Points

The following extensions are defined for PAP v1.0. Core extensions (Sections 13.1--13.4) were introduced in v0.4. Recovery mandates (Section 13.5), TEE attestation (Section 13.6), and payment proof validation (Section 13.7) were added in v0.7. All extensions are OPTIONAL; a conformant implementation MAY support none, some, or all of them.

13.1. Payment Proof

A mandate MAY carry a payment_proof field containing a zero-knowledge payment commitment. PAP does not define the payment protocol; it defines the integration point. Only cryptographic commitments are stored — *never* amounts, destinations, mints, or other identifying payment data.

The PaymentProof type is a tagged enum with two variants:

Variant	Inner Type	Description
Lightning	Bolt11Hash	SHA-256 of a BOLT-11 invoice payment hash
Ecash	CashuTokenHash	SHA-256 of a Cashu blind-signed token

Table 27

13.1.1. 13.1.1. Bolt11Hash

A commitment to a Lightning Network payment. The hash field contains the base64url-no-pad encoded SHA-256 of the BOLT-11 invoice payment hash. The preimage is never stored.

```
{
  "type": "Lightning",
  "hash": "<base64url-no-pad SHA-256>"
}
```

13.1.2. 13.1.2. CashuTokenHash

A commitment to a Cashu ecash token. The hash field contains the base64url-no-pad encoded SHA-256 of the blind-signed token. The token itself is never stored.

```
{
  "type": "Ecash",
  "hash": "<base64url-no-pad SHA-256>"
}
```

13.1.3. 13.1.3. Payment Proof Properties

* The proof contains *only* a cryptographic commitment hash.
* No amounts, destinations, mints, or routing data are stored.
* The vendor MUST NOT be able to identify the payer from the proof.
* The proof MUST be unlinkable to the principal's identity.
* The payment proof is included in the mandate's canonical form for signing.
* If a mandate's scope includes schema:PayAction, a payment proof SHOULD be attached. Implementations MAY reject mandates that permit payment actions without a proof.

13.1.4. 13.1.4. Ecash Blind Signature Protocol

PAP includes a reference implementation of the Chaumian blind signature scheme in the pap-ecash crate. The scheme uses RFC 9474 RSABSSA-SHA384-PSS (non-augmented variant, randomize = false).

Protocol parameters:

Parameter	Value
Scheme	RSABSSA-SHA384-PSS (RFC 9474 § 4.2, non-augmented)
Key size	2048 bits (production); 1024 bits (tests only)
Commitment	SHA-256(serial ‖ signature), base64url-no-pad
Serial size	32 bytes, randomly chosen by the client

Table 28

Protocol steps:

1. *Request* — client calls ecash_request(serial, mint_pk). Returns a BlindToken containing a randomly-blinded serial. Only the blinded_message() bytes are transmitted to the mint.
2. *Mint* — mint calls ecash_mint_sign(blinded_msg, keypair). Returns raw blind-signature bytes to the client.
3. *Unblind* — client calls ecash_unblind(blind_token, blind_sig, mint_pk). Returns the spendable EcashToken { serial, signature }.
4. *Attach* — client calls token.to_payment_proof() and includes the result in the mandate's payment_proof field.
5. *Verify* — payee calls ecash_verify(token, mint_pk). Valid tokens have a correct RSA-PSS signature over serial.
6. *Redeem* — payee calls ecash_redeem(token, mint_pk, registry). Atomically verifies and records serial in the spent registry, preventing double-spend.

Unlinkability invariant: The random blinding factor applied in step 1 means that the blinded_message bytes transmitted to the mint are statistically independent of the final (serial, signature) pair. The mint cannot link a signing operation to a subsequent redemption.

Double-spend invariant: ecash_redeem MUST return EcashError::DoubleSpend on any second call with the same serial, regardless of signature validity.

Test vectors: The conformance test suite is in crates/pap-ecash/. Run the following to generate and verify all test vectors:

```
cargo test -p pap-ecash -- --nocapture
```

The ecash_test_vector test uses a freshly-generated 1024-bit test key (test-only size) and serial 0x000…001 (32 bytes). Because blind-rsa-signatures v0.14 uses OsRng internally (no injectable RNG), the blinding factor and PSS salt are non-deterministic. The test therefore validates structural invariants (correct verification, 43-char base64url commitment) rather than pinning an exact byte value.

C FFI: pap_ecash_mint_keypair_generate, pap_ecash_blind, pap_ecash_blind_message_bytes, pap_ecash_mint_sign, pap_ecash_unblind, pap_ecash_verify, pap_ecash_spent_registry_new, pap_ecash_redeem, pap_ecash_token_payment_proof_commitment.

WASM: EcashMintKeypair, EcashBlindToken, EcashToken, ecashMintSign, ecashVerify.

13.2. Payment Proof Verification

A receiving agent that requires payment MUST: 1. Extract the payment_proof from the mandate. 2. Validate the proof's structural integrity (valid base64url, 32-byte SHA-256 commitment). 3. Verify the proof against the payment network (out of band): - *Lightning*: verify the BOLT-11 payment hash preimage - *Ecash*: verify the Cashu token with the issuing mint 4. Accept or reject the session based on verification.

13.2.1. 13.2.1. Receipt Payment Proof Commitment

When a transaction receipt is created for a schema:PayAction, the receipt MUST include a payment_proof_commitment field containing the commitment hash from the mandate's payment proof. This enables auditing without revealing payment details.

A receipt validator MUST check: 1. The payment_proof_commitment is present for payment actions. 2. The commitment matches the mandate's payment proof commitment. 3. The commitment is included in the receipt's canonical form for co-signing.

The verification protocol between the receiving agent and the payment network is out of scope for this specification.

13.3. Continuity Tokens

A continuity token enables stateful relationships across sessions without requiring the vendor to retain state.

Field	Type	Required	Description
schema_type	String	REQUIRED	Schema.org type describing the encrypted payload shape
vendor_did	String	REQUIRED	DID of the vendor that issued this token
encrypted_payload	String	REQUIRED	Vendor-encrypted state (opaque to orchestrator)
ttl	DateTime	REQUIRED	Expiry timestamp, set by the principal
issued_at	DateTime	REQUIRED	Issuance timestamp

Table 29

13.3.1. 13.3.1. Continuity Token Lifecycle

1. At session close, the vendor encrypts its internal state and returns it as a continuity token to the orchestrator.
2. The orchestrator stores the token locally. The vendor retains nothing.
3. When the principal returns, the orchestrator presents the token to the vendor.
4. The vendor decrypts the payload and resumes the relationship.
5. The principal controls the TTL. The vendor MUST NOT set or extend the TTL.
6. To sever the relationship, the principal deletes the token. No revocation notice is required.

13.3.2. 13.3.2. Continuity Token Properties

* The schema_type MUST be inspectable by the orchestrator without decrypting the payload.
* The vendor MUST NOT be able to write to the continuity token without the principal presenting it.

* The encrypted payload format is vendor-defined and opaque to the protocol.

13.4. Auto-Approval Policies

An auto-approval policy allows the principal to pre-authorize certain categories of actions without per-transaction approval.

Field	Type	Required	Description
name	String	REQUIRED	Human-readable policy name
scope	Scope	REQUIRED	Subset of the mandate scope this policy applies to
max_value	Number or null	OPTIONAL	Maximum transaction value for auto-approval (currency-agnostic)
zero_additional_disclosure	Boolean	REQUIRED	If true, auto-approve only when zero additional disclosure is required beyond the mandate
authored_at	DateTime	REQUIRED	Timestamp when the principal authored this policy

Table 30

13.4.1. 13.4.1. Auto-Approval Constraints

* The policy scope MUST be contained by the mandate scope (Section 5.4.5). A policy MUST NOT be more permissive than the mandate.
* Policies are principal-authored and orchestrator-enforced. An agent MUST NOT trigger a policy change by requesting it.
* zero_additional_disclosure defaults to true. When true, the orchestrator MUST auto-approve only when the agent’s disclosure requirements are fully covered by the existing mandate.

* If max_value is set and the transaction value exceeds it, the orchestrator MUST request explicit principal approval.

13.5. M-of-N Social Recovery

Principal identity recovery via a designated notary quorum. No central recovery authority. The principal designates N notary DIDs at setup time; any M co-signers from that set can authorize key rotation.

13.5.1. 13.5.1. Recovery Mandate

A principal creates a RecoveryMandate while they still control their key, designating the notary set and threshold.

Field	Type	Required	Description
principal_did	String	REQUIRED	DID of the principal creating the mandate
threshold	Integer	REQUIRED	M: minimum co-signatures required (1 M N)
notary_dids	Array<String>	REQUIRED	N designated notary DIDs (no duplicates)
created_at	DateTime	REQUIRED	Mandate creation timestamp
signature	String	REQUIRED	Ed25519 signature by the principal

Table 31

Constraints: - threshold MUST be 1 and notary_dids.length. - notary_dids MUST NOT contain duplicate entries. - The mandate MUST be signed by the principal’s current key. - Only one recovery mandate per principal DID. A new mandate replaces any previous one.

13.5.2. 13.5.2. Recovery Request

When recovery is needed, a RecoveryRequest is created identifying the old principal, the new principal keypair, and the authorizing recovery mandate.

Field	Type	Required	Description
old_principal_did	String	REQUIRED	DID of the principal being recovered
new_principal_did	String	REQUIRED	DID of the new principal keypair
recovery_mandate_hash	String	REQUIRED	SHA-256 hash of the authorizing RecoveryMandate
requested_at	DateTime	REQUIRED	Request timestamp

Table 32

The canonical bytes of the recovery request are the message that each notary signs independently.

13.5.3. 13.5.3. Partial Recovery Signature (Blind)

Each notary signs the recovery request independently. Notaries MUST NOT communicate with each other during recovery — they learn nothing about which other notaries have been contacted (threshold blind signature scheme).

Before signing, a notary MUST verify: 1. The recovery mandate was signed by the old principal. 2. The notary’s own DID is in the designated notary set. 3. The request references the correct recovery mandate hash. 4. The request’s old_principal_did matches the mandate.

Field	Type	Required	Description
notary_did	String	REQUIRED	DID of the signing notary
signature	String	REQUIRED	Ed25519 signature over the RecoveryRequest canonical bytes
signed_at	DateTime	REQUIRED	Timestamp of the notary’s signature

Table 33

13.5.4. 13.5.4. Recovery Proof Assembly

A recovery coordinator collects M partial signatures and assembles a RecoveryProof. Verification of the proof MUST check:

1. The recovery mandate was signed by the old principal.
2. At least M partial signatures are present.
3. All signers are in the designated notary set.
4. No duplicate signers.
5. All partial signatures are cryptographically valid.

13.5.5. 13.5.5. Revocation Proof and Broadcast

After successful recovery, a RevocationProof is created and broadcast to federation peers.

Field	Type	Required	Description
old_principal_did	String	REQUIRED	The revoked DID
new_principal_did	String	REQUIRED	The replacement DID
recovery_proof_hash	String	REQUIRED	SHA-256 hash of the RecoveryProof
revoked_at	DateTime	REQUIRED	Revocation timestamp
signature	String	REQUIRED	Ed25519 signature by the new principal key

Table 34

The revocation proof MUST be signed by the new principal key (proving possession). Federation peers that receive a valid revocation MUST: - Mark the old principal DID as revoked. - Reject any future operations using the old DID. - Remove the old recovery mandate from their NotarySet.

13.5.6. 13.5.6. NotarySet Registry

Each federation node maintains a NotarySet — a registry of recovery mandates queryable by principal DID. The NotarySet: - Stores signed recovery mandates. - Tracks revoked principal DIDs. - Rejects mandate registration for already-revoked DIDs. - Processes revocation broadcasts from federation peers.

13.5.7. 13.5.7. Security Properties

* *No central authority.* Recovery requires M independent notaries.
* *Blind co-signing.* Notaries do not learn which other notaries participate in a recovery event.
* *Old key revocation.* The old principal DID is cryptographically revoked and broadcast to all federation peers.
* *Notary set immutability.* The notary set is fixed at mandate creation time by the principal. It cannot be modified without creating a new mandate signed by the principal.
* *Threshold enforcement.* Fewer than M signatures MUST be rejected. Duplicate signers MUST be rejected.

13.6. TEE Attestation

A mandate or session MAY carry a Trusted Execution Environment (TEE) attestation to provide evidence that an agent is executing within an isolated enclave. TEE attestation is OPTIONAL and does NOT elevate a TEE to equivalence with local trust (Section 3.4).

13.6.1. 13.6.1. Attestation Object

Field	Type	Required	Description
enclave_type	String	REQUIRED	TEE platform identifier (e.g., "sgx", "sev-snp", "trustzone")
measurement	String	REQUIRED	Enclave measurement hash (base64url-no-pad)
attestation_report	String	REQUIRED	Platform-specific attestation report (base64url-no-pad)
timestamp	DateTime	REQUIRED	Attestation generation timestamp (RFC 3339)
nonce	String	REQUIRED	Challenge nonce binding this attestation to the current session (UUID v4)

Table 35

13.6.2. 13.6.2. Attestation Verification

A verifier MUST:

1. Verify the attestation_report against the TEE platform’s root of trust (platform-specific, out of scope).
2. Verify that measurement matches an expected enclave binary hash (implementation-defined allowlist).
3. Verify that nonce matches the session’s challenge nonce.

4. Verify that timestamp is within an acceptable window (implementations SHOULD reject attestations older than 60 seconds).

13.6.3. 13.6.3. Trust Boundaries

TEE attestation provides evidence of code integrity, not behavioral correctness. Specifically:

* A TEE attestation MUST NOT be treated as equivalent to a mandate. An agent in a TEE still requires a valid mandate chain.
* A TEE attestation MUST NOT be used to expand scope beyond what the mandate permits.
* The principal MAY use TEE attestation as an input to auto-approval policies (Section 13.4) but MUST NOT be required to accept TEE attestation as a substitute for consent.

13.6.4. 13.6.4. Implementation Notes

The reference implementation provides TEE attestation support via the pap-tee crate, which is compiled only when opted into as a dependency. Integration with pap-core is gated behind the tee Cargo feature flag.

* *pap-tee crate*: Defines AttestationEvidence, EnclaveType, the AttestationVerifier trait, and a SoftwareSimulator for integration testing without hardware.
* *pap-core tee feature*: Adds an optional attestation field to Session and provides open_with_attestation().
* *ProtocolMessage::TokenAccepted*: Carries an optional attestation field as opaque JSON (serde_json::Value). Receivers parse it via AttestationEvidence::from_value().

The SoftwareSimulator uses EnclaveType::Software and signs attestation reports with an Ed25519 key. It is intended for conformance testing (Appendix D, tests E-13 through E-15) and MUST NOT be deployed in production.

13.7. Payment Proof Validation

Section 13.1 defines the payment proof integration point. This section specifies the validation requirements that a conformant implementation MUST satisfy when payment proofs are present.

13.7.1. 13.7.1. Proof Format Registry

PAP defines the following payment proof format prefixes:

Prefix	Protocol	Description
ecash:blind:v1:	Chaumian ecash	Blind-signed mint tokens (Section 13.1)
ln:preimage:v1:	Lightning Network	Hash preimage proof of payment
zk:receipt:v1:	Zero-knowledge proof	ZK proof of value transfer

Table 36

Implementations MAY support additional formats using the pap:payment: namespace prefix.

13.7.2. 13.7.2. Validation Requirements

A receiving agent that requires payment MUST:

1. Parse the payment_proof field and identify the format prefix.
2. If the format is not supported, reject the mandate with a PaymentFormatUnsupported error.
3. Verify the proof against the appropriate payment backend (mint, Lightning node, or ZK verifier). The verification protocol is out of scope for this specification.
4. Verify that the proof amount meets the agent’s minimum requirement for the requested action.
5. Verify that the proof has not been previously consumed (double-spend protection).

13.7.3. 13.7.3. Privacy Requirements

* Payment proof verification MUST NOT reveal the payer’s identity to the payment backend.
* The receiving agent MUST NOT store payment proofs beyond the session duration unless required by applicable law.
* Payment proofs MUST NOT appear in transaction receipts (Section 11.5).

13.8. Chat and Real-Time Communication

13.8.1. 13.8.1. Overview

PAP provides a natural foundation for zero-trust, privacy-preserving real-time communication between principals. A personal agent MAY advertise schema:CommunicateAction in the federation registry — exactly as a service agent advertises schema:SearchAction. This makes a principal discoverable for chat without requiring a phone number, email address, or centrally-administered identity. Discoverability is opt-in, scoped, and revocable through the standard mandate system.

Chat is not a new protocol. It is the *Phase 4 streaming extension* of the standard 6-phase handshake, applied to a schema:CommunicateAction session.

13.8.2. 13.8.2. Capability Grant

A CapabilityToken scoped to schema:CommunicateAction MUST be issued by the initiating principal (or a delegated orchestrator) and signed with a principal key. The token:

* MUST set action = "schema:CommunicateAction".
* MUST set target_did to the receiving principal’s agent DID.
* MAY set a ttl appropriate for the conversation duration.
* MAY carry a scope restricting the permitted communication modes (e.g., text-only, text+audio, text+audio+video).

13.8.3. 13.8.3. Phase 4 Streaming Mode

After Phase 3 (disclosure), instead of a single task execution, the session transitions to *streaming mode*:

1. *Phase 4 execute* (client → server, no payload): the receiving agent returns ExecutionResult containing a schema:Conversation JSON-LD object. This signals that the session SHOULD remain open.

2. *StreamingMessage frames* (bidirectional, Phase 4): either party MAY send StreamingMessage frames carrying DIDComm basicmessage protocol payloads (see Section 13.8.5). Each frame MUST include:

   * id: a UUID for ack correlation.
   * content: a JSON object conforming to the DIDComm basicmessage body schema.

3. *StreamingAck* (responding side): upon receiving a StreamingMessage, the server MUST reply with either a StreamingAck (delivery confirmed) or a StreamingMessage (reply).

4. The session MUST remain open until either party sends SessionClose (Phase 6). Implementations SHOULD NOT proceed to Phase 5 (receipt co-signing) until the conversation is concluded.

13.8.4. 13.8.4. Message Format (DIDComm basicmessage)

The content field of each StreamingMessage MUST conform to the DIDComm basicmessage 2.0 protocol body:

```
{
  "type": "https://didcomm.org/basicmessage/2.0/message",
  "id": "<UUID>",
  "body": {
    "content": "<text of message>"
  }
}
```

The DIDComm wrapping (plaintext, signed, or encrypted) is applied at the Envelope layer via PapToDIDComm (Section 5.6). For chat sessions, implementations SHOULD use at minimum DIDCommSigned to bind each message to the sender's session DID.

13.8.5. 13.8.5. Receipt

Upon SessionClose, the receipt (Phase 5) MUST record:

* action = "schema:CommunicateAction"
* executed: a summary string, e.g., "schema:Conversation".
* disclosed_by_initiator / disclosed_by_receiver: property references only (e.g., ["schema:name"]). Message *content* MUST NOT appear in receipts.
* Both parties' session DIDs as initiating_agent_did / receiving_agent_did (ephemeral, unlinked from principal DIDs).

13.8.6. 13.8.6. Group Chat Rooms

A group chat room is an agent with its own DID that implements AgentHandler and maintains one session per member:

* The room DID is registered in the federation with capability: ["schema:CommunicateAction"].
* The room owner issues a separate CapabilityToken to each member, all targeting the room DID.
* Each member runs the standard 6-phase handshake against the room DID. After Phase 4 (streaming mode open), the room agent fans out each StreamingMessage to all other connected members.

* Group membership is enforced by the token system: only principals holding a valid token may connect. Revocation follows the standard mandate revocation flow (Section 8).
* Rooms MAY be hosted locally (Papillon instance) or on any federation peer. A room hosted on a federation peer is discoverable via its DID advertisement.

13.8.7. 13.8.7. Audio and Video

Audio and video calls follow the same pattern as text chat, using WebRTC as the media transport:

1. PAP Phases 14 establish identity, authorization, and streaming mode. The CapabilityToken scope SHOULD include the permitted media types (e.g., text+audio+video).
2. *SDP negotiation* is carried via StreamingMessage frames: the offerer sends a StreamingMessage whose content.body contains the SDP offer; the answerer replies with SDP answer. ICE candidates are exchanged as subsequent frames.
3. WebRTC DTLS-SRTP establishes the media channel out-of-band. PAP does not inspect or relay media.
4. Implementations MAY route ICE/TURN through an OHTTP relay to conceal participant IP addresses.
5. The PAP receipt records call metadata (duration, participant session DIDs, permitted media scope) but MUST NOT include audio or video content.

13.8.8. 13.8.8. Privacy Properties

Chat sessions inherit all PAP privacy guarantees:

* *Ephemeral session DIDs* — neither party’s principal DID appears in message frames or SDP.
* *OHTTP relay* — IP addresses hidden from the relay operator.
* *Receipts* — property references only; no message content.
* *Discoverability* — controlled by the principal’s federation advertisement; opt-in.
* *Forward secrecy* — DIDComm anoncrypt (ECDH-ES + A256GCM) MAY be applied to StreamingMessage content for per-message forward secrecy.

14. Transport Binding

14.1. HTTP/JSON Transport

PAP defines an HTTP/JSON transport binding for the 6-phase handshake. This binding is the default transport for PAP v1.0. Implementations MAY define additional transport bindings.

14.2. Agent Server Endpoints

A receiving agent MUST expose the following HTTP endpoints:

Method	Path	Phase	Request	Response
POST	/session	1	TokenPresentation	TokenAccepted or TokenRejected
POST	/session/{id}/did	2	SessionDidExchange	SessionDidAck
POST	/session/{id}/disclosure	3	DisclosureOffer	DisclosureAccepted
POST	/session/{id}/execute	4	(empty body)	ExecutionResult
POST	/session/{id}/receipt	5	ReceiptForCoSign	ReceiptCoSigned
POST	/session/{id}/close	6	SessionClose	SessionClosed

Table 37

The {id} path parameter is the session ID returned in Phase 1.

14.3. Agent Handler Interface

Implementations MUST implement a handler interface with the following operations:

Operation	Phase	Input	Output
handle_token	1	CapabilityToken	(session_id, receiver_session_did)
handle_did_exchange	2	session_id, initiator_session_did	()
handle_disclosure	3	session_id, disclosures	()
execute	4	session_id	JSON result
co_sign_receipt	5	TransactionReceipt	TransactionReceipt (co-signed)
handle_close	6	session_id	()

Table 38

14.4. Endpoint Resolution

Endpoint resolution maps a DID to a transport endpoint URL. In production, this SHOULD be backed by DID Document service endpoints. Implementations MAY use in-memory registries for development and testing.

14.5. Content Type

All HTTP request and response bodies MUST use Content-Type: application/json. Implementations SHOULD set Accept: application/json on requests.

14.6. Error Handling

If a phase handler returns an error, the server MUST respond with HTTP status 500 and a ProtocolMessage::Error payload containing a code and message.

If the request body does not match the expected message type for the endpoint, the server MUST respond with HTTP status 400.

14.7. WebSocket Transport

Implementations MAY support a WebSocket transport binding as an alternative to the HTTP/JSON binding. The WebSocket binding is OPTIONAL and provides full-duplex communication for sessions that benefit from lower-latency message exchange.

14.7.1. 14.7.1. Connection Lifecycle

1. The initiating agent opens a WebSocket connection to the receiving agent's WebSocket endpoint.
2. All 6 phases of the session handshake (Section 6.3) are conducted as JSON messages over the WebSocket connection.
3. Each message MUST be a JSON-serialized Envelope (Section 8.2).
4. The connection MUST be closed after Phase 6 (session close).

14.7.2. 14.7.2. Endpoint Format

A WebSocket endpoint MUST use the wss:// scheme. Implementations MUST NOT use unencrypted ws:// connections in production.

The endpoint URL MUST be published in the agent's DID Document service array with type set to "PAPWebSocket":

```
{
  "id": "did:key:z...#pap-ws",
  "type": "PAPWebSocket",
  "serviceEndpoint": "wss://agent.example.com/pap/ws"
}
```

14.7.3. 14.7.3. Message Framing

Each WebSocket text frame MUST contain exactly one JSON-serialized Envelope. Binary frames MUST NOT be used. Implementations MUST reject connections that send binary frames.

14.7.4. 14.7.4. Sequence Enforcement

Envelope sequence number rules (Section 8.2.2) apply identically over WebSocket. Out-of-order messages MUST be rejected.

14.8. Oblivious HTTP (OHTTP) Transport

Implementations MAY support Oblivious HTTP [RFC 9458] as a transport binding. OHTTP provides request unlinkability at the network layer, preventing the receiving agent's operator from correlating requests by IP address.

14.8.1. 14.8.1. Architecture

An OHTTP deployment interposes a relay between the initiating agent and the receiving agent:

Initiator -> OHTTP Relay -> Receiving Agent (Gateway)

The relay sees the initiator’s IP but not the request content. The receiving agent sees the request content but not the initiator’s IP.

14.8.2. 14.8.2. Encapsulation

Each PAP protocol message MUST be encapsulated as an OHTTP Binary HTTP request targeting the corresponding HTTP/JSON endpoint (Section 14.2). The Content-Type MUST remain application/json.

14.8.3. 14.8.3. Key Configuration

The receiving agent MUST publish its OHTTP key configuration in its DID Document service array with type set to "PAPObliviousHTTP":

```
{
  "id": "did:key:z...#pap-ohttp",
  "type": "PAPObliviousHTTP",
  "serviceEndpoint": "https://agent.example.com/pap/ohttp",
  "ohttpKeyConfig": "<base64url-encoded-key-config>"
}
```

14.8.4. 14.8.4. Relay Selection

The initiating agent selects the OHTTP relay. The relay MUST NOT be operated by the same entity as the receiving agent. The protocol does not define relay discovery; implementations SHOULD allow the principal to configure trusted relays.

14.9. DIDComm Transport

Implementations MAY support DIDComm Messaging v2 [DIDCOMM-V2] as a transport binding. DIDComm provides authenticated encryption at the message layer, enabling transport-independent secure messaging between agents identified by DIDs.

14.9.1. 14.9.1. Message Mapping

Each PAP protocol message (Section 8.1) MUST be wrapped in a DIDComm plaintext message with the following mapping:

DIDComm Field	Value
type	https://pap.dev/protocol/1.0/{message_type}
from	Sender’s DID (session DID after Phase 2)
to	Array containing recipient’s DID
body	The PAP protocol message payload
created_time	Envelope timestamp (Unix epoch seconds)

Table 39

Where {message_type} is the lowercase, hyphenated form of the PAP message type (e.g., token-presentation, session-did-exchange).

14.9.2. 14.9.2. Encryption

DIDComm messages MUST use authenticated encryption (authcrypt) after Phase 2 when both session DIDs are known. Phase 1 messages MAY use anonymous encryption (anoncrypt) since the initiator’s session DID is not yet established.

14.9.3. 14.9.3. Service Endpoint

A DIDComm-capable agent MUST publish a DIDComm service endpoint in its DID Document:

```
{
  "id": "did:key:z...#pap-didcomm",
  "type": "DIDCommMessaging",
  "serviceEndpoint": {
    "uri": "https://agent.example.com/pap/didcomm",
    "accept": ["didcomm/v2"]
  }
}
```

14.10. Transport Negotiation

When an agent supports multiple transport bindings, the initiating agent MUST select a transport by inspecting the receiving agent’s DID Document service array. The preference order SHOULD be:

1. OHTTP (strongest privacy properties)
2. DIDComm (authenticated encryption at message layer)

3. WebSocket (lower latency for interactive sessions)
4. HTTP/JSON (default, widest compatibility)

If the receiving agent's DID Document contains no service entries, the initiating agent MUST fall back to HTTP/JSON with endpoint resolution (Section 14.4).

14.11. DIDComm v2 Envelope Compatibility

PAP defines an optional DIDComm v2 envelope compatibility layer that wraps PAP protocol envelopes inside DIDComm v2 message formats. This allows PAP agents to interoperate with DIDComm-native agents without changing the PAP protocol itself. This section specifies the detailed wire formats used by the DIDComm transport binding (Section 14.9).

14.11.1. 14.11.1. Design Principles

* PAP mandate, session, and receipt semantics are fully preserved.
* Only the outer transport envelope changes; the inner PAP Envelope (including its Ed25519 signature) travels intact inside the DIDComm message body.
* The DIDComm layer provides additional transport-level integrity (JWS) or confidentiality (JWE) on top of PAP's own signatures.
* This is a shim — existing pap-transport behavior is unaffected.

14.11.2. 14.11.2. Plaintext Messages

A PAP envelope is wrapped in a DIDComm v2 plaintext message:

```
{
  "id": "<uuid>",
  "typ": "application/didcomm-plain+json",
  "type": "https://pap.baur.dev/proto/1.0/<message-slug>",
  "from": "<sender-did>",
  "to": ["<recipient-did>"],
  "created_time": <unix-timestamp>,
  "body": { <full PAP Envelope as JSON> }
}
```

The type field uses PAP message type URIs under the namespace https://pap.baur.dev/proto/1.0/, with kebab-case slugs derived from the ProtocolMessage variant name (e.g., session-did-ack, execution-result, token-presentation).

The body field contains the complete PAP Envelope including its signature field, so the receiving agent can verify the PAP-level signature independently of the DIDComm layer.

14.11.3. 14.11.3. Signed Messages (Ed25519 JWS)

A signed DIDComm v2 message uses JWS General JSON Serialization (RFC 7515) with the EdDSA algorithm (RFC 8037):

```
{
  "payload": "<base64url(plaintext-json)>",
  "signatures": [{
    "protected": "<base64url({\"typ\":\"application/didcomm-signed+json\",\"alg\":\"Ed
DSA\"})>",
    "signature": "<base64url(Ed25519-signature)>"
  }]
}
```

The signing input is ASCII(protected) || '.' || ASCII(payload) where both values are base64url-encoded without padding (RFC 4648 Section 5). The signature is computed with Ed25519 (RFC 8032).

Verifiers MUST reject messages where: - The alg header value is not "EdDSA". - The signature does not verify against the expected key. - The decoded payload is not valid DIDComm v2 plaintext JSON.

14.11.4. 14.11.4. Encrypted Messages (ECDH-ES + A256GCM JWE)

An encrypted DIDComm v2 message uses JWE JSON Serialization with anonymous encryption (anoncrypt):

- *Key Agreement*: ECDH-ES (direct, no key wrapping) via X25519 Diffie-Hellman. The sender generates an ephemeral X25519 keypair. The recipient's Ed25519 public key is converted to X25519 using the Edwards-to-Montgomery birational map.
- *Key Derivation*: Concat KDF (NIST SP 800-56A Section 5.8.1) with algId = "A256GCM", empty apu, and apv = SHA-256(recipient-did).
- *Content Encryption*: AES-256-GCM with a random 96-bit IV. The base64url-encoded protected header serves as Additional Authenticated Data (AAD).

```
{
  "protected": "<base64url(header-json)>",
  "recipients": [{
    "header": { "kid": "<recipient-did>" },
    "encrypted_key": ""
  }],
  "iv": "<base64url(96-bit-nonce)>",
  "ciphertext": "<base64url(aes-gcm-ciphertext)>",
  "tag": "<base64url(128-bit-auth-tag)>"
}
```

The protected header contains:

Field	Value
typ	"application/didcomm-encrypted+json"
alg	"ECDH-ES"
enc	"A256GCM"
epk	{"kty":"OKP","crv":"X25519","x":"<base64url-pubkey>"}
apv	"<base64url(SHA-256(recipient-did))>"

Table 40

The encrypted_key field is empty for ECDH-ES direct key agreement (the content encryption key is derived directly from the shared secret).

14.11.5. 14.11.5. Ed25519 to X25519 Key Conversion

DIDComm v2 encryption requires X25519 keys for key agreement. PAP agents use Ed25519 keys (via did:key). The conversion is:

* *Public key*: Decompress the Ed25519 compressed Edwards Y coordinate, then apply the Edwards-to-Montgomery birational map to obtain the X25519 public key (32 bytes).
* *Private key*: Compute SHA-512(Ed25519-seed)[0..32]. The X25519 library applies standard clamping (clear bits 0-2, clear bit 255, set bit 254).

This conversion is consistent: the X25519 public key derived from the converted private key matches the X25519 public key derived from the original Ed25519 public key.

14.11.6. 14.11.6. Translation Rules

Direction	Operation
PAP → DIDComm Plaintext	Serialize PAP Envelope into DIDComm body
PAP → DIDComm Signed	Build plaintext, then apply Ed25519 JWS
PAP → DIDComm Encrypted	Build plaintext, then apply

		ECDH-ES + A256GCM JWE
DIDComm Plaintext → PAP	Deserialize body field as PAP Envelope	
DIDComm Signed → PAP	Verify JWS, then extract PAP Envelope from body	
DIDComm Encrypted → PAP	Decrypt JWE, then extract PAP Envelope from body	

Table 41

In all cases, the PAP Envelope.signature field (if present) remains intact and can be verified independently using the session's ephemeral key.

15. PAP URI Scheme

15.1. Overview

The pap URI scheme identifies agents, capabilities, and resources within the Principal Agent Protocol. A pap:// URI is always an expression of *intent* — resolving one initiates a PAP mandate-scoped interaction, not a raw network request.

The scheme family consists of three variants:

Scheme	Meaning
pap://	PAP-native transport; client negotiates protocol
pap+https://	PAP mandate scope applied over HTTPS transport
pap+wss://	PAP mandate scope applied over WebSocket transport

Table 42

pap+https:// and pap+wss:// are *recapture schemes*. They apply PAP semantics — mandate enforcement, selective disclosure, co-signed receipts — to existing transports. The remote endpoint does not need to implement PAP. The client enforces the protocol locally. A pap+https:// URI is still an HTTPS request under the hood; the principal's mandate scope wraps it regardless of whether the server is PAP-aware.

15.2. Syntax

```
pap-uri         = pap-scheme "://" pap-authority pap-path [ "?" pap-query ]

pap-scheme      = "pap" / "pap+https" / "pap+wss"

pap-authority   = registry-host / did-authority / catalog-name

registry-host   = host [ ":" port ]
                  ; authority is the hostname only; agent slug appears in path

did-authority   = "did:key:" base58-multicodec-key
                  ; PAP parsers MUST treat "did:key:" as an atomic authority
                  ; token. Standard RFC 3986 host parsing (which disallows
                  ; colons) MUST NOT be applied to did-authority. A PAP URI
                  ; parser identifies did-authority by the "did:key:" prefix
                  ; before applying any other rule.

catalog-name    = 1*( ALPHA / DIGIT / "-" / "_" )
                  ; MUST NOT be a reserved word (receipt, canvas, settings)
                  ; resolved against local catalog before dispatch

pap-path        = registry-path / simple-path

registry-path   = "/agents/" agent-slug "/" schema-action-type
simple-path     = "/" schema-action-type
                  ; Schema.org action type, e.g. "SearchAction"

pap-query       = pap-param *( "&" pap-param )
pap-param       = schema-property "=" pap-value
                  ; values MUST be percent-encoded per RFC 3986 §2.1
                  ; "+" MUST NOT be used as a space encoding in pap-query
```

Examples:

```
; Networked agent via Chrysalis registry
pap://chrysalis.example.com/agents/arxiv/SearchAction?query=quantum%20computing

; Direct peer-to-peer via DID (no registry)
pap://did:key:z6MkhaXgBZDvotDkL5257faiztiGiC2QtKLGpbnnEGta2doK/SearchAction

; Local catalog shorthand — resolved before dispatch
pap://arxiv/SearchAction?query=quantum%20computing
pap://wikipedia/ReadAction?name=Rust%20programming

; Receipt deep-link
pap://receipt/RCP_abc123

; Recapture schemes — PAP scope over existing transports
pap+https://api.example.com/agents/bookings/BuyAction?offer=flight-abc
pap+wss://stream.example.com/agents/feed/ListenAction
```

15.3. Resolution

A conforming client MUST resolve a pap:// URI using the following priority chain, in order:

1. *Special authority* — if the authority is one of the reserved words (receipt, canvas, settings), resolve locally without any network lookup. See §15.7. Do not proceed to subsequent steps.

2. *did:key: authority* — if the authority begins with did:key:, resolve directly via DID Document endpoint discovery (Section 14.4). No registry lookup. Initiates a PAP handshake with the identified agent.

3. *Catalog name* — if the authority contains no . character and does not begin with did:, the client MUST check its local agent catalog for an entry whose name field matches the authority (case-insensitive). If found, rewrite the URI to the agent's registered DID and resolve via step 1.

4. *Registry hostname* — if the authority contains a . character, or matches localhost, or is a valid IPv4 address or IPv6 literal, treat it as a Chrysalis registry host. Resolve by querying the registry's /agents/{slug}/ routes (Section 14.1) using the path-embedded agent slug, and initiate a PAP handshake with the returned agent endpoint. The . heuristic MUST NOT be applied to localhost or IP literals; they are always treated as registry hosts.

If resolution fails at all steps, the client MUST render an inline error in place of the activated link, showing the unresolved URI and a human-readable explanation. The client MUST NOT navigate away from the current canvas or dismiss existing content. The client MUST NOT silently fall back to a raw HTTP request.

15.4. Action Type and Query Parameters

The action type path segment MUST be a Schema.org action type (e.g. SearchAction, BuyAction, ReadAction). For registry URIs the full path is /agents/{slug}/{ActionType}; for catalog and DID URIs the path is /{ActionType}. Clients SHOULD use the action type to pre-filter agents during resolution — if a catalog agent does not advertise the requested action type in its capability array, it MUST NOT be selected.

Query parameters MUST use Schema.org property names as keys. Values MUST be percent-encoded per RFC 3986 §2.1; + MUST NOT be used as a space encoding. Clients MAY pass query parameters directly to the agent as the intent payload. Agents MAY ignore unknown parameters.

15.5. Recapture Semantics (pap+https://, pap+wss://)

When a pap+https:// or pap+wss:// URI is resolved:

1. The active mandate scope MUST be checked before the request is made. If no mandate is in scope, the client MUST NOT proceed.

2. The request is made over the underlying transport (HTTPS or WSS) with the standard PAP session headers included where the server accepts them.

3. The client MUST record what was disclosed and generate a receipt entry regardless of whether the server participates in the PAP handshake.

4. The remote endpoint’s response is treated as agent output and rendered via the standard block renderer pipeline.

For pap+wss:// URIs, the connection lifecycle (establishment, keepalive, and termination) follows the mandate-scoped session lifecycle defined in §5. Streaming-specific semantics (chunked responses, event framing) are deferred to v1.1.

This allows principals to bring existing web services under PAP governance without requiring those services to be modified.

v1.0 scope note: In v1.0, pap+https:// and pap+wss:// URIs are parsed and classified by conforming clients. Full mandate enforcement (steps 14 above) requires the mandate enforcement layer, which is deferred to a post-v1.0 milestone. v1.0 clients MUST NOT silently downgrade a pap+https:// URI to an unscoped HTTPS request. They MUST either enforce the mandate or reject the request with a clear principal-visible error explaining that recapture enforcement is not yet available.

15.6. Link Rendering

Any string value in a JSON-LD agent response that begins with pap://, pap+https://, or pap+wss:// MUST be rendered as a navigable link by conforming clients. Activating such a link MUST dispatch the URI as intent through the same pipeline as a principal-typed query — it is not a browser navigation event.

This enables agent-rendered content to form a navigable graph of intent-links without requiring any special page routing. Every link is a new PAP interaction.

Agent-rendered link security: Clients MUST visually distinguish links originating from agent-rendered content from links typed directly by the principal. Before dispatching an agent-rendered pap:// link, clients SHOULD display the full URI and the identity of the agent that produced it, and require explicit principal confirmation. This prevents injection attacks where a malicious or compromised agent response induces the client to execute unintended actions.

Agent-rendered links MUST NOT activate the settings, canvas, or receipt special authorities (§ 15.7). Clients MUST silently reject such links and MAY log the attempt for principal review.

15.7. Special Authorities

The following authority values are reserved and MUST be handled by the client without registry or catalog lookup:

Authority	Meaning
receipt	Deep-link to a receipt by session ID. pap://receipt/{session-id} opens the receipt detail view.
canvas	Deep-link to a canvas block. pap://canvas/{canvas-id}/{block-id} navigates to the referenced block.
settings	Opens the settings panel. pap://settings/{tab} opens a specific tab.

Table 43

16. Security Considerations

16.1. Cryptographic Algorithms

PAP v1.0 uses exclusively:

* *Ed25519* (RFC 8032) for all signatures.
* *SHA-256* (FIPS 180-4) for all hashes.
* *Base64url without padding* (RFC 4648 Section 5) for all binary-to-text encoding.
* *Base58btc* for DID key encoding.

Implementations MUST use these algorithms for PAP v1.0. All signable structures carry a SignatureAlgorithm field (serialized as the JWS alg string, e.g. "EdDSA") to enable forward-compatible algorithm negotiation. The field defaults to Ed25519 when absent. Implementations MUST reject algorithms they do not support. The did:key multicodec prefix encodes the algorithm of the public key.

Future versions of this specification MAY introduce additional algorithms (e.g., ML-DSA-65 for post-quantum resistance).

16.2. Key Management

* Principal private keys SHOULD be stored in hardware security modules or platform authenticators (WebAuthn). They MUST NOT be stored in plaintext in configuration files or environment variables in production.
* Session private keys MUST be held only in memory for the duration of the session. They MUST NOT be persisted to disk.

* Signing keys for agent operators (used to sign advertisements) SHOULD be protected with access controls appropriate to the deployment environment.

16.3. Nonce Management

* Capability token nonces MUST be stored in a consumed-nonce set for at least the duration of the token's validity period.
* Implementations SHOULD periodically purge expired nonces to prevent unbounded growth of the consumed-nonce set.
* If a receiver restarts and loses its consumed-nonce set, it SHOULD reject all tokens issued before the restart by comparing issued_at against its restart timestamp.

16.4. Replay Protection

Multiple layers provide replay protection:

1. *Token nonces:* Each capability token has a UUID v4 nonce consumed on first use.
2. *Envelope sequencing:* Sequence numbers are monotonically increasing within a session. Out-of-order envelopes MUST be rejected.
3. *Token expiry:* Tokens carry an expires_at timestamp. Expired tokens MUST be rejected.
4. *Session ephemerality:* Session keys are discarded at close. A replayed session message cannot be verified against the original session keys.

16.5. Denial of Service

* Implementations SHOULD rate-limit token presentation requests to prevent resource exhaustion from session initiation floods.
* Federation sync operations SHOULD be rate-limited per peer.
* Marketplace registries SHOULD limit the number of advertisements per operator DID.

16.6. Man-in-the-Middle

* After Phase 2 (DID exchange), all envelopes MUST be signed by the sender's session key. An attacker who intercepts envelopes cannot forge valid signatures without the session private key.
* The initial token presentation (Phase 1) is protected by the orchestrator's signature on the capability token. An attacker cannot forge a valid token without the orchestrator's private key.
* Implementations SHOULD use TLS for all HTTP transport to protect against passive eavesdropping.

16.7. Context Leakage

* The DisclosureOffer (Phase 3) MUST contain only SD-JWT disclosures permitted by the mandate's disclosure set.
* The orchestrator MUST verify that the agent's requires_disclosure is satisfiable by the mandate before issuing a capability token. An agent MUST NOT receive a token if its disclosure requirements exceed the principal's authorization.
* Receipts MUST NOT contain personal data values (Section 11.5).

16.8. Mandate Chain Depth

Implementations SHOULD enforce a maximum mandate chain depth to prevent resource exhaustion during chain verification. A maximum depth of 10 is RECOMMENDED.

16.9. Clock Skew

* Implementations MUST use UTC for all timestamps.
* Implementations SHOULD tolerate clock skew of up to 30 seconds for token expiry and mandate TTL checks.
* Implementations MAY use NTP or similar time synchronization protocols to minimize skew.

16.10. Canonical JSON Determinism

The security of mandate hashing and signature verification depends on deterministic JSON serialization. Implementations MUST ensure that the canonical JSON form produces identical bytes for the same logical content.

Implementations SHOULD: - Use a JSON serializer that produces consistent key ordering. - Represent numbers without unnecessary precision. - Use RFC 3339 with explicit UTC offset for all timestamps.

If an implementation cannot guarantee deterministic JSON output, it MUST use an alternative canonical form (e.g., JCS [RFC 8785]) and document the choice.

16.11. Attack Surface Summary

Attack Vector	Mitigation	Spec Section
Context profiling	Ephemeral session DIDs	4.4, 6.3.2
Over-disclosure	SD-JWT structural binding +	7, 9.3

	marketplace filtering	
Replay attacks	Nonce consumption + envelope sequencing	6.2.2, 8.2.2
Delegation bypass	Scope containment + TTL bounds	5.4.5, 5.5
Mandate tampering	Parent hash + signature chain	5.3, 5.6
Platform lock-in	Federated discovery, no central registry	10
Payment linkability	ZK commitments (Lightning BOLT-11, Cashu ecash)	13.1
Session correlation	Session keys discarded at close	4.4, 6.3.6
Stale authorization	Decay state machine + non-renewal revocation	5.7
Advertisement spoofing	Signed advertisements, registry rejects unsigned	9.4
Retention violation	TEE attestation for no_retention sessions	5.4.4.1, 13.6
Vouch ring / Sybil peers	Vouch budget + age requirement + diverse paths	10.7.3
Metric-based ranking capture	Anti-ranking requirement on marketplace queries	9.6

Table 44

Appendix A. Example: Zero-Disclosure Search

This appendix illustrates a complete PAP transaction with zero personal disclosure.

A.1. A.1. Setup

```
Principal generates keypair -> did:key:zPrincipal
Orchestrator keypair -> did:key:zOrch
Search agent operator keypair -> did:key:zSearch
```

A.2. A.2. Root Mandate

```
{
  "principal_did": "did:key:zPrincipal",
  "agent_did": "did:key:zOrch",
  "issuer_did": "did:key:zPrincipal",
  "parent_mandate_hash": null,
  "scope": {
    "actions": [{"action": "schema:SearchAction"}]
  },
  "disclosure_set": {"entries": []},
  "ttl": "2026-03-15T20:00:00+00:00",
  "decay_state": "Active",
  "issued_at": "2026-03-15T16:00:00+00:00",
  "payment_proof": null,
  "signature": "<base64url>"
}
```

A.3. A.3. Marketplace Query

```
query_satisfiable("schema:SearchAction", available=[])
  -> [SearchAgent] (requires_disclosure: [])
  -> Filtered out: agents requiring personal disclosure
```

A.4. A.4. Session Handshake

```
Phase 1: Orchestrator -> SearchAgent: TokenPresentation
         SearchAgent -> Orchestrator: TokenAccepted(session_id, recv_did)

Phase 2: Orchestrator -> SearchAgent: SessionDidExchange(init_did)
         SearchAgent -> Orchestrator: SessionDidAck

Phase 3: Orchestrator -> SearchAgent: DisclosureOffer([])
         SearchAgent -> Orchestrator: DisclosureAccepted

Phase 4: SearchAgent -> Orchestrator: ExecutionResult({...})

Phase 5: Orchestrator -> SearchAgent: ReceiptForCoSign(receipt)
         SearchAgent -> Orchestrator: ReceiptCoSigned(receipt)

Phase 6: Orchestrator -> SearchAgent: SessionClose
         SearchAgent -> Orchestrator: SessionClosed
```

A.5. A.5. Receipt

```
{
  "session_id": "<uuid>",
  "action": "schema:SearchAction",
  "initiating_agent_did": "did:key:zInitSess",
  "receiving_agent_did": "did:key:zRecvSess",
  "disclosed_by_initiator": [],
  "disclosed_by_receiver": ["operator:search_executed"],
  "executed": "schema:SearchAction executed",
  "returned": "schema:SearchResult returned",
  "timestamp": "2026-03-15T16:05:00+00:00",
  "signatures": ["<initiator_sig>", "<receiver_sig>"]
}
```

Zero personal properties disclosed. Both session DIDs are ephemeral and discarded. The receipt is auditable but contains no personal data.

Appendix B. Example: Selective Disclosure Flight Booking

B.1. B.1. Disclosure Set

```
{
  "entries": [{
    "type": "schema:Person",
    "permitted_properties": ["schema:name", "schema:nationality"],
    "prohibited_properties": ["schema:email", "schema:telephone"],
    "session_only": true,
    "no_retention": true
  }]
}
```

B.2. B.2. SD-JWT Claims

```
Claims: {name: "Alice", email: "alice@example.com",
         nationality: "US", telephone: "+1-555-0100"}
Disclosed: [name, nationality]
Withheld: [email, telephone]  (cryptographically uncommitted)
```

B.3. B.3. Marketplace Filtering

```
SkyBook Flight Agent:  requires [name, nationality]    -> satisfiable
LuxAir Premium Agent:  requires [name, nationality, email] -> FILTERED OUT
StayWell Hotel Agent:  wrong object type               -> not matched
```

B.4. B.4. Receipt

```
{
  "disclosed_by_initiator": [
    "schema:Person.schema:name",
    "schema:Person.schema:nationality"
  ],
  "disclosed_by_receiver": ["operator:booking_confirmed"]
}
```

Values "Alice" and "US" never appear in the receipt.

Appendix C. Example: 4-Level Delegation Chain

Level 0: Principal (root of trust)
Level 1: Orchestrator
  scope: [Search, Reserve(Flight), Reserve(Lodging), Pay]
  ttl: 4h

Level 2: Trip Planner (delegated from Orchestrator)
  scope: [Search, Reserve(Flight)] (subset of Level 1)
  ttl: 3h (< 4h)
  parent_mandate_hash: hash(Level 1 mandate)

Level 3: Booking Agent (delegated from Trip Planner)
  scope: [Reserve(Flight)] (subset of Level 2)
  ttl: 2h (< 3h)
  parent_mandate_hash: hash(Level 2 mandate)

Attempted violations: - Booking Agent delegates PayAction ->
DelegationExceedsScope - Booking Agent delegates with TTL > 2h ->
DelegationExceedsTtl

Chain verification: verify_chain([principal_key, orch_key,
planner_key])

Appendix D. Conformance Test Matrix

A conformant PAP v1.0 implementation MUST pass all tests in the *Core* category. Tests in the *Extension* category apply only when the implementation supports the corresponding extension.

D.1. D.1. Core Protocol Tests

ID	Test	Spec Section	Requirement
C-01	Root mandate sign and verify	5.2	MUST

C-02	Mandate hash determinism (same input produces same hash)	5.3	MUST
C-03	Scope containment: child subset of parent accepted	5.4.5	MUST
C-04	Scope containment: child exceeding parent rejected	5.4.5, 5.5 R1	MUST
C-05	Scope containment: child broadening object constraint rejected	5.4.5	MUST
C-06	Delegation TTL: child TTL <= parent TTL accepted	5.5 R2	MUST
C-07	Delegation TTL: child TTL > parent TTL rejected	5.5 R2	MUST
C-08	Parent hash binding: correct hash accepted	5.5 R3	MUST
C-09	Parent hash binding: incorrect hash rejected	5.5 R3	MUST
C-10	Issuer chain: child issuer_did == parent agent_did	5.5 R4	MUST
C-11	Principal propagation: child principal_did == parent principal_did	5.5 R5	MUST
C-12	Root mandate: parent_mandate_hash is null	5.5 R6	MUST
C-13	Mandate chain verification: 2-level chain	5.6	MUST
C-14	Mandate chain verification: 3-level chain	5.6	MUST
C-15	Mandate chain verification: invalid signature in chain rejected	5.6	MUST
C-16	Decay state: Active within TTL	5.7	MUST
C-17	Decay state: Degraded within	5.7	MUST

	decay window			
C-18	Decay state: ReadOnly after TTL expiry	5.7	MUST	
C-19	Decay state: Suspended is terminal (no renewal)	5.7.1	MUST	
C-20	Decay state: invalid transition rejected	5.7.1	MUST	
C-21	Capability token sign and verify	6.2	MUST	
C-22	Capability token: wrong target_did rejected	6.2.2	MUST	
C-23	Capability token: nonce replay rejected	6.2.2	MUST	
C-24	Capability token: expired token rejected	6.2.2	MUST	
C-25	Session state machine: Initiated -> Open -> Executed -> Closed	6.1	MUST	
C-26	Session state machine: invalid transition rejected	6.1	MUST	
C-27	Session state machine: early termination from Initiated	6.1	MUST	
C-28	SD-JWT commitment and disclosure verification	7.4, 7.5	MUST	
C-29	SD-JWT: disclosure hash not in commitment rejected	7.5	MUST	
C-30	SD-JWT: unsigned commitment rejected	7.4	MUST	
C-31	SD-JWT: zero-disclosure session accepted	7.6	MUST	
C-32	SD-JWT: partial disclosure (subset of claims)	7.3	MUST	

C-33	Envelope sign and verify with session keys	8.2.1	MUST
C-34	Envelope: wrong key verification fails	8.2.2	MUST
C-35	Envelope: out-of-sequence rejected	8.2.2	MUST
C-36	Envelope: tampered payload detected	8.2.1	MUST
C-37	Receipt: co-signed by both parties	11.3	MUST
C-38	Receipt: contains property references, not values	11.5	MUST
C-39	Receipt: zero-disclosure receipt valid	11.5	MUST
C-40	Receipt: wrong key co-sign verification fails	11.4	MUST
C-41	Advertisement: unsigned advertisement rejected by registry	9.4	MUST
C-42	Advertisement: content hash deduplication	9.5, 10.5	MUST
C-43	Marketplace: query by action returns matching agents	9.3	MUST
C-44	Marketplace: disclosure satisfiability filtering	9.3	MUST
C-45	VC envelope: wrap and unwrap mandate	12.2	MUST
C-46	VC envelope: unsigned VC rejected	12.3	MUST
C-47	Session: no_retention disclosure rejected without TEE attestation	5.4.4.1	MUST
C-48	Session attestation: sign and	11.6	MUST

	verify bilateral attestation		
C-49	Session attestation: per-action-type segmentation enforced	11.6	MUST

Table 45

D.2. D.2. Transport Tests

ID	Test	Spec Section	Requirement
T-01	HTTP/JSON: full 6-phase handshake over HTTP	14.2	MUST
T-02	HTTP/JSON: error response with code and message	14.6	MUST
T-03	HTTP/JSON: wrong message type returns 400	14.6	MUST
T-04	WebSocket: full 6-phase handshake over WebSocket	14.7	OPTIONAL
T-05	WebSocket: binary frame rejected	14.7.3	OPTIONAL
T-06	OHTTP: encapsulated request reaches gateway	14.8	OPTIONAL
T-07	DIDComm: message mapping roundtrip	14.9.1	OPTIONAL

Table 46

D.3. D.3. Extension Tests

ID	Test	Spec Section	Requirement
E-01	Payment proof: mandate with valid proof accepted	13.1, 13.7	OPTIONAL
E-02	Payment proof: unsupported	13.7.2	OPTIONAL

	format rejected		
E-03	Payment proof: double-spend rejected	13.7.2	OPTIONAL
E-04	Continuity token: creation and expiry check	13.3	OPTIONAL
E-05	Continuity token: expired token rejected	13.3.1	OPTIONAL
E-06	Continuity token: principal-controlled TTL	13.3.2	OPTIONAL
E-07	Auto-approval: policy within mandate scope accepted	13.4	OPTIONAL
E-08	Auto-approval: policy exceeding mandate rejected	13.4.1	OPTIONAL
E-09	Auto-approval: transaction exceeding max_value requires approval	13.4.1	OPTIONAL
E-10	Recovery mandate: pap:RecoverAction in scope	13.5.1	OPTIONAL
E-11	Recovery mandate: delegation attempt rejected	13.5.3	OPTIONAL
E-12	Recovery mandate: short TTL enforced	13.5.3	OPTIONAL
E-13	TEE attestation: valid attestation with matching nonce	13.6.2	OPTIONAL
E-14	TEE attestation: stale attestation rejected	13.6.2	OPTIONAL
E-15	TEE attestation: does not expand mandate scope	13.6.3	OPTIONAL
E-16	Marketplace: query results not ranked by operator metrics	9.6	MUST
E-17	Marketplace: operator	9.6	MUST

		metrics excluded from content hash		

Table 47

D.4. D.4. Federation Tests

ID	Test	Spec Section	Requirement
F-01	Federation: QueryByAction returns matching advertisements	10.3, 10.4	MUST
F-02	Federation: Announce and AnnounceAck roundtrip	10.3, 10.4	MUST
F-03	Federation: content-hash deduplication on merge	10.5	MUST
F-04	Federation: unsigned advertisement skipped on merge	10.5	MUST
F-05	Federation: transitive peer discovery	10.6	OPTIONAL
F-06	Federation: peer registration requires minimum vouches	10.7.2	SHOULD
F-07	Federation: vouch budget enforced (max 3/year)	10.7.3	SHOULD
F-08	Federation: probationary peer cannot vouch	10.7.3	SHOULD
F-09	Federation: vouch signature verification	10.7.2	MUST

Table 48

D.5. D.5. Trust Invariant Summary

A conformant implementation MUST demonstrate all eight trust invariants hold:

#	Invariant	Key Tests
TI-1	Mandate scope is cryptographically bounded	C-03, C-04, C-05
TI-2	Session DIDs are ephemeral and unlinkable to principal	C-25, C-27
TI-3	Receipts contain property references, never values	C-37, C-38, C-39
TI-4	Delegation chains enforce depth and TTL bounds	C-06, C-07, C-13, C-14
TI-5	Decay states follow the defined state machine	C-16, C-17, C-18, C-19, C-20
TI-6	no_retention requires TEE attestation	C-47
TI-7	Marketplace queries are ranking-free	E-16, E-17
TI-8	Peer vouching enforces budget and age constraints	F-06, F-07, F-08

Table 49

End of specification.

Appendix E. References

E.1. Normative References

[RFC 2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC 8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, May 2017.

[RFC 3339] Klyne, G. and C. Newman, "Date and Time on the Internet: Timestamps", RFC 3339, July 2002.

[RFC 4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, October 2006.

[RFC 8032] Josefsson, S. and I. Liusvaara, "Edwards-Curve Digital Signature Algorithm (EdDSA)", RFC 8032, January 2017.

[DID-CORE] Sporny, M., Guy, A., Sabadello, M., and D. Reed, "Decentralized Identifiers (DIDs) v1.0", W3C Recommendation, July 2022.

[DID-KEY] Longley, D. and M. Sporny, "The did:key Method v0.7", W3C Community Group Report.

[SD-JWT-08] Fett, D., Yasuda, K., and B. Campbell, "Selective Disclosure for JWTs (SD-JWT)", Internet-Draft draft-ietf-oauth-selective-disclosure-jwt-08.

[VC-DATA-MODEL-2.0] Sporny, M., et al., "Verifiable Credentials Data Model v2.0", W3C Recommendation.

[WEBAUTHN] Balfanz, D., et al., "Web Authentication: An API for accessing Public Key Credentials Level 2", W3C Recommendation.

E.2. Informative References

[RFC 8785] Rundgren, A., Jordan, B., and S. Erdtman, "JSON Canonicalization Scheme (JCS)", RFC 8785, June 2020.

[RFC 9458] Thomson, M. and C. A. Wood, "Oblivious HTTP", RFC 9458, January 2024.

[DIDCOMM-V2] Curren, S., Looker, T., and O. Terbu, "DIDComm Messaging v2.0", Decentralized Identity Foundation, 2022.

[RFC 6455] Fette, I. and A. Melnikov, "The WebSocket Protocol", RFC 6455, December 2011.

Appendix F. IANA and Vocabulary References

F.1. Schema.org Vocabulary

PAP uses Schema.org (https://schema.org (https://schema.org)) as the vocabulary for action types, object types, and property references. The following Schema.org types are referenced in this specification:

Action Types: - schema:SearchAction -- Search for information - schema:ReserveAction -- Reserve a resource (flight, hotel, etc.) - schema:PayAction -- Make a payment - schema:CheckAction -- Check a condition or status - schema:ReadAction -- Read a resource

Object Types: - schema:Flight -- A flight - schema:Lodging -- Lodging accommodation - schema:WebPage -- A web page

Entity Types: - schema:Person -- A person - schema:Organization -- An organization - schema:Service -- A service - schema:Order -- An order - schema:Subscription -- A subscription

Property References: - schema:name -- Name of a person or entity - schema:email -- Email address - schema:telephone -- Phone number - schema:nationality -- Nationality

Implementations MAY use additional Schema.org types and properties. Implementations MAY define additional namespaced vocabularies using a prefix notation (e.g., custom:MyAction). Custom vocabularies SHOULD be documented.

F.2. W3C Standards

Standard	URI	Usage
DID Core 1.0	https://www.w3.org/TR/did-core/ (https://www.w3.org/TR/did-core/)	DID document structure
DID Key Method	https://w3c-ccg.github.io/did-method-key/ (https://w3c-ccg.github.io/did-method-key/)	did:key derivation
VC Data Model 2.0	https://www.w3.org/TR/vc-data-model-2.0/ (https://www.w3.org/TR/vc-data-model-2.0/)	Credential envelope

Table 50

F.3. IETF Standards

Standard	RFC/Draft	Usage
RFC 2119	Key words	Requirement levels
RFC 8174	Key words update	Requirement levels clarification
RFC 3339	Date and Time on the Internet	Timestamp format

RFC 4648	Base Encodings	Base64url encoding
RFC 8032	Edwards-Curve Digital Signature Algorithm	Ed25519 signatures
RFC 8785	JSON Canonicalization Scheme	Canonical JSON (RECOMMENDED)
RFC 9458	Oblivious HTTP	OHTTP transport binding (Section 14.8)
draft-ietf-oauth-selective-disclosure-jwt-08	SD-JWT	Selective disclosure

Table 51

F.4. WebAuthn

Standard	URI	Usage
Web Authentication Level 2	https://www.w3.org/TR/webauthn-2/ (https://www.w3.org/TR/webauthn-2/)	Device-bound key generation

Table 52

F.5. Multicodec

The Ed25519 public key multicodec prefix is 0xed01 as registered in the Multicodec table (https://github.com/multiformats/multicodec (https://github.com/multiformats/multicodec)).

F.6. Reserved Namespace Prefixes

Prefix	Namespace	Authority
schema:	https://schema.org (https://schema.org)	Schema.org Community
operator:	Implementation-defined	Agent operator

pap:	Reserved for PAP extensions	PAP specification

Table 53

Appendix G. Changelog

G.1. v1.0 (2026-03-24)

* Promoted specification from Draft to Approved status.
* *Section 13.5:* Added recovery mandate extension with recovery proof binding and short-TTL constraints.
* *Section 13.6:* Added TEE attestation extension with enclave measurement verification and trust boundary rules.
* *Section 13.7:* Added payment proof validation requirements including format registry, double-spend protection, and privacy constraints.
* *Section 14.7:* Added WebSocket transport binding with connection lifecycle, message framing, and sequence enforcement.
* *Section 14.8:* Added Oblivious HTTP (OHTTP) transport binding with relay architecture and key configuration.
* *Section 14.9:* Added DIDComm v2 transport binding with message mapping and authenticated encryption.
* *Section 14.10:* Added transport negotiation rules with privacy-preference ordering.
* *Appendix D:* Added conformance test matrix.
* Updated all version references from v0.1 to v1.0.
* Added DIDComm and WebSocket to normative/informative references.

G.2. v0.7 (2026-03-10)

* Added recovery mandate extension (Section 13.5).
* Added TEE attestation extension (Section 13.6).
* Added payment proof format registry and validation (Section 13.7).

G.3. v0.6 (2026-02-28)

* Added WebSocket transport binding (Section 14.7).
* Added OHTTP transport binding (Section 14.8).
* Added DIDComm transport binding (Section 14.9).
* Added transport negotiation (Section 14.10).

G.4. v0.4 (2026-02-01)

* Initial public draft with core protocol:
  - Trust model and threat model (Section 3).
  - Identity layer with did:key (Section 4).

- Mandate structure and delegation rules (Section 5).
- Session lifecycle with 6-phase handshake (Section 6).
- SD-JWT disclosure protocol (Section 7).
- Protocol messages and envelope (Section 8).
- Marketplace advertisement schema (Section 9).
- Federation protocol (Section 10).
- Receipt format (Section 11).
- Verifiable Credential envelope (Section 12).
- Payment proof integration point (Section 13.1).
- Continuity tokens (Section 13.3).
- Auto-approval policies (Section 13.4).
- HTTP/JSON transport binding (Section 14.1--14.6).

Author’s Address

Todd Baur
Baur Software
Email: todd@baursoftware.com