Individual Submission
Internet-Draft
Intended status: Standards Track
Expires: 20 October 2026

G. Valverde
Zentity

18 April 2026

PACT: Private Agent Consent and Trust Profile for OAuth 2.1 and CIBA
draft-valverde-oauth-pact-00

Abstract

PACT (Private Agent Consent and Trust Profile) is a security profile of OAuth 2.1 for privacy-preserving agent delegation. It extends VEIL and composes OIDC CIBA Core 1.0 and OAuth Token Exchange (RFC 8693). PACT defines a durable-host plus ephemeral-session control plane, a delegation token claim vocabulary, runtime identity proofs using Ed25519 JWTs, capability grants with typed constraints and usage limits, risk-graduated consent routing, and claim narrowing on token exchange to non-agent audiences.

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 20 October 2026.

Copyright Notice

Table of Contents

1. Introduction

An agent acting on behalf of a human needs three properties at once: machine authentication (the caller proves it is a specific runtime, not merely the application that launched it), human consent (the human approves sensitive actions through a channel the agent cannot subvert), and privacy-preserving delegation (the relying party learns who acted without receiving a globally trackable identifier).

Existing specifications cover each property in isolation. CIBA [CIBA-Core] provides a backchannel consent channel. DPoP [RFC9449] sender-constrains tokens. RAR [RFC9396] carries structured authorization payloads. Token Exchange [RFC8693] rebinds audience and scope. PACT specifies how agent identity, human consent, and token issuance compose into a single delegation flow.

The profile composes and constrains the following specifications:

Concern	Specifications
Secure Transport	OAuth 2.1 [I-D.ietf-oauth-v2-1], PKCE [RFC7636], PAR [RFC9126], DPoP [RFC9449]
Structured Intent	Rich Authorization Requests [RFC9396]
Agent Control Plane	PACT (this document), OAuth Client Attestation [I-D.ietf-oauth-attestation-based-client-auth]
Consent Channel	CIBA [CIBA-Core]
Token Semantics	Token Exchange [RFC8693], Token Introspection [RFC7662], OIDC Core Pairwise Identifiers [OIDC-Core]

Table 1

This document specifies: pairwise agent identifiers (Section 4.3), risk-graduated consent routing (Section 6), usage-limited capability grants (Section 5.5), cryptographic binding chains (Section 11), ephemeral identity disclosure, and claim narrowing on token exchange (Section 8).

1.1. Relation to Other Work

Several concurrent individual submissions address overlapping parts of the agent-delegation problem. The Agent Auth Protocol [AAP] and [I-D.aap-oauth-profile] define a JWT claim vocabulary for agent-acting-as-user semantics; PACT aligns with their claim names where they coincide. [I-D.oauth-ai-agents-on-behalf-of-user] explores the subject model when an agent acts on behalf of an authenticated user; PACT's session identity and act.sub derivation build on the same premise. The Agent-to-Agent Protocol [A2A] covers agent-to-agent discovery and card exchange, which complements the consent and token profile defined here. PACT's scope is narrower than any of these: it specifies CIBA binding, typed-constraint capability grants, and claim narrowing on token exchange to non-agent audiences.

2. Conventions and Terminology

2.1. Notational Conventions

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 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

2.2. Versioning

PACT uses MAJOR.MINOR versioning, with -draft appended for pre-release revisions. Implementations MUST reject configurations or discovery documents with an unrecognized major version. Implementations SHOULD accept documents with a higher minor version than expected and ignore unrecognized fields, preserving forward compatibility.

2.3. Definitions

Agent Session: An ephemeral runtime identity for one live process. Each session holds its own Ed25519 keypair. The private key exists only in process memory.

Agent-Assertion: A short-lived EdDSA JWT (typ: agent-assertion+jwt) signed by the session private key, proving possession of the runtime identity at the time of a consent request.

Approval Strength: A capability-level declaration of the minimum consent mechanism required: none (auto-approve), session (active user interaction), or biometric (unforgeable user verification).

Authorization Server (AS): The server that implements this profile, managing identity registration, capability grants, consent routing, and token issuance.

Binding Chain: The sequence of cryptographic proofs connecting an OAuth user token through host registration, session registration, runtime assertion, human consent, and delegated token issuance.

Capability: A server-defined named action with optional JSON Schema for input and output, and a required approval strength. PACT's unit of authorization.

Capability Grant: An authorization record linking one agent session to one capability, with optional typed constraints, usage limits, and an expiration time.

Client: The process that holds the host identity, manages keys, signs JWTs, and presents tools to the agent runtime.

Constraint: A typed restriction on a grant's input parameters. Operators: eq, min, max, in, not_in.

Durable Host: The persistent installation identity for a client environment. Survives across process restarts. Holds an Ed25519 keypair persisted to disk.

Host Policy: A durable capability grant attached to a host rather than a session. Survives session expiry and seeds new sessions.

Pairwise Agent Identifier: A per-relying-party pseudonym derived from the session ID and the RP's sector identifier, preventing cross-RP agent correlation.

Relying Party (RP): A service that receives and validates delegated agent tokens.

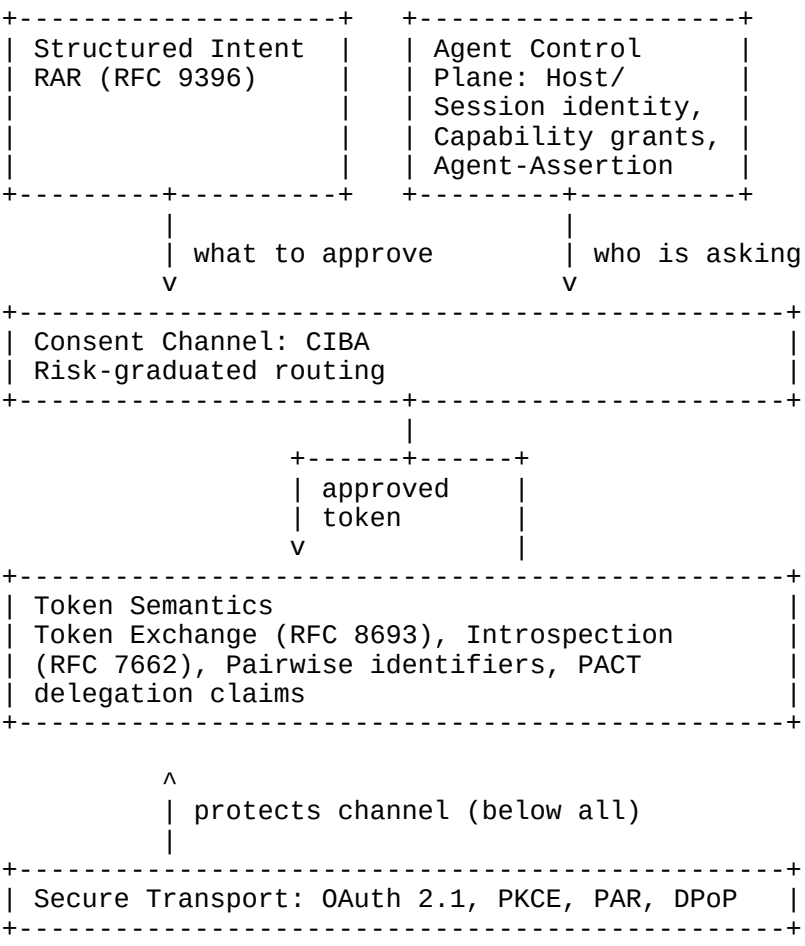
Session Grant: An ephemeral capability grant attached to one agent session. Seeded from host policies at registration; may be elevated via consent.

Usage Ledger: An append-only record of capability executions, used to enforce daily limits and cooldowns atomically.

3. Compositional Architecture

3.1. Five Compositional Concerns

PACT's constituent standards group into five concerns: secure transport, structured intent, agent control plane, consent channel, and token semantics. Each concern produces inputs for the next, from machine authentication through human approval to token issuance and relying-party verification.



Secure Transport sits beneath the other four concerns because removing any transport spec changes security properties, not the agent model. DPoP is the one transport spec that crosses into agent territory: when token exchange mints a downstream token from a CIBA access token, DPoP re-binds the issued token to the caller's proof-of-possession key.

Structured Intent carries typed payloads through the flow. RAR (authorization_details) is submitted on the CIBA request, retained on the CIBA request record during consent evaluation, and projected onto downstream exchanged tokens (Section 8). The approved typed payload does not appear on the CIBA-issued access token itself.

Agent Control Plane establishes who is asking. It produces two outputs: the Agent-Assertion JWT that enters the consent channel as runtime proof, and the capability grants that determine whether consent can short-circuit into automatic approval.

Consent Channel carries the human approval interaction. CIBA binds the runtime proof to a specific consent request. The CIBA auth_req_id serves as the trace identifier correlating agent identity, consent, and token issuance.

Token Semantics encodes the authorized delegation. Token exchange rebinds audience and recomputes pairwise identifiers. Introspection re-evaluates session lifecycle at query time.

3.2. Runtime Participants

Role	Function
Agent	The AI actor scoped to a conversation, task, or session. Does not hold keys directly.
Client	Process holding the host identity; manages keys, signs JWTs, presents tools.
Authorization Server	Manages registrations, capability grants, consent routing, and token issuance.
Relying Party	Receives delegated tokens; validates agent identity via introspection or JWT verification.
User	The human principal who approves sensitive actions via CIBA.

Table 2

3.3. Three Caller Classes

PACT distinguishes three caller classes by authentication mode and scope.

Caller	Authentication	Scope
Browser user	Session cookie	Dashboard and browser-only surfaces
User-delegated machine	OAuth access token exchanged into a dedicated DPoP-bound bootstrap token	Agent host/session registration, revocation
Pure machine client	client_credentials access token	Introspection, resource-server APIs

Table 3

Registration, introspection, and token exchange are machine-facing OAuth surfaces; human consent is obtained separately through CIBA (Section 6). A client bootstrapping an agent host MUST NOT reuse a login token for registration: it MUST first exchange that token for a DPoP-bound bootstrap token carrying narrow agent scopes (Section 4.1).

4. Principal Separation

PACT separates agent identity into three layers with distinct lifetimes and audiences. The host is durable (persistent across process restarts). The session is ephemeral (fresh per runtime instance, enabling per-session audit). The pairwise identifier shown to each relying party is uncorrelatable across services.

4.1. Durable Host

A host is the persistent installation identity for a client environment. It represents a specific installation on a specific machine, not a user or an application.

Bootstrap. The client first exchanges its pairwise login token via Token Exchange [RFC8693] for a short-lived DPoP-bound bootstrap token carrying agent:host.register. POST {host_registration_endpoint} authenticates with that bootstrap token, not the login token.

Identity properties:

Property	Value
Key type	Ed25519 [RFC8037]
Identity anchor	JWK Thumbprint [RFC7638], SHA-256
Persistence	Client-side file, server-side record
Uniqueness	One thumbprint per (user, client) pair. Same user and client MAY have multiple hosts (different machines).
Binding	A host key MUST NOT be rebound across users or OAuth clients.

Table 4

Key storage. The client MUST persist the host keypair in a namespace derived from the server URL, OAuth client ID, and the authenticated account subject (or another stable per-user identifier):

```
~/.zentity/hosts/
  <SHA-256(serverUrl + ":" + clientId + ":" + accountSub)>.json
```

The file MUST be stored with mode 0600. The directory MUST be created with mode 0700.

Attestation. Host registration MAY include vendor attestation headers per [I-D.ietf-oauth-attestation-based-client-auth]:

* OAuth-Client-Attestation: a JWT signed by a trusted vendor, containing a cnf.jwk matching the host’s public key.

* OAuth-Client-Attestation-PoP: a proof-of-possession JWT signed by the host’s private key.

The server verifies attestation JWTs against a pre-configured set of trusted issuer JWKS URLs. Verification uses a hardened JWKS fetcher that rejects unsafe remote key sources (private/loopback IPs, non-HTTPS in production, responses exceeding a fixed size cap, long timeouts).

Attestation results in an elevated trust tier that widens default host policy (Section 5.4).

4.2. Ephemeral Session

An agent session is the runtime identity for one live process. Each session holds its own Ed25519 keypair. The private key MUST exist only in process memory, and MUST NOT be persisted to disk.

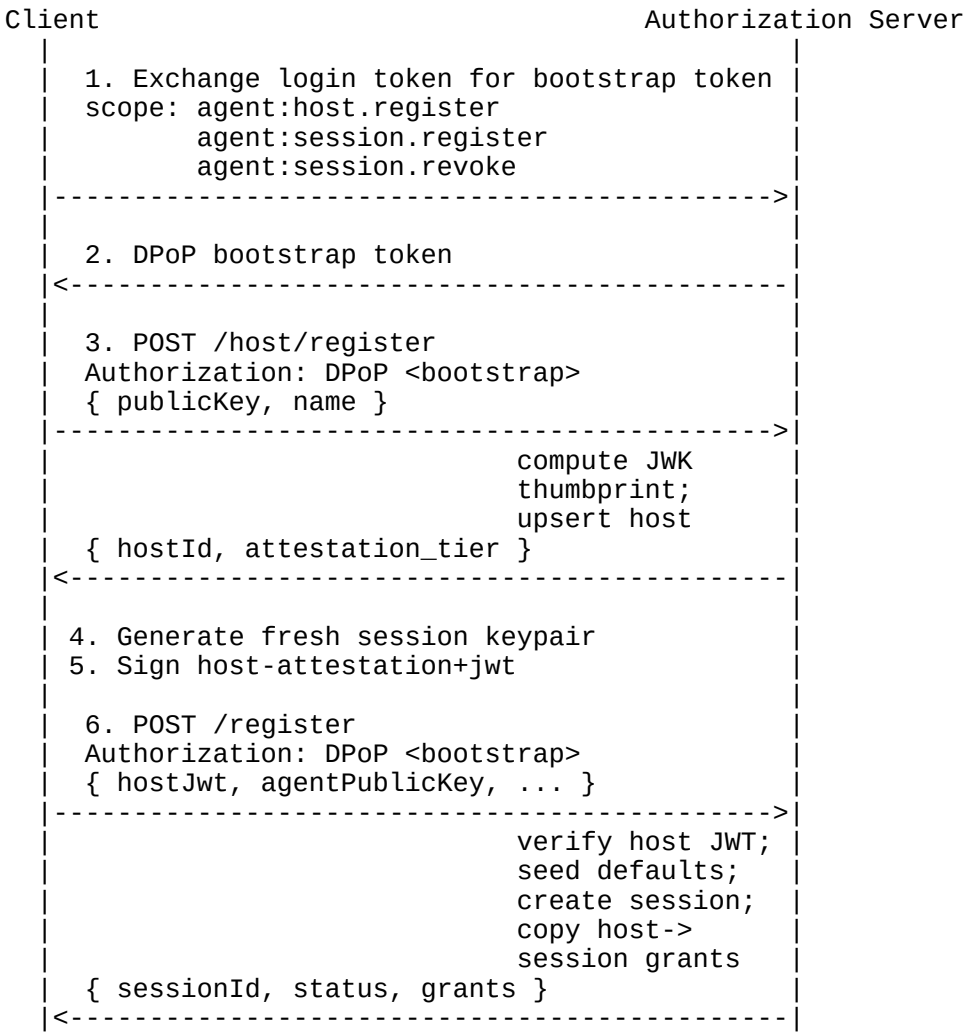
Registration. POST {registration_endpoint} with the bootstrap token carrying agent:session.register and a host attestation JWT.

The host attestation JWT (typ: host-attestation+jwt) proves the client possesses the durable host key:

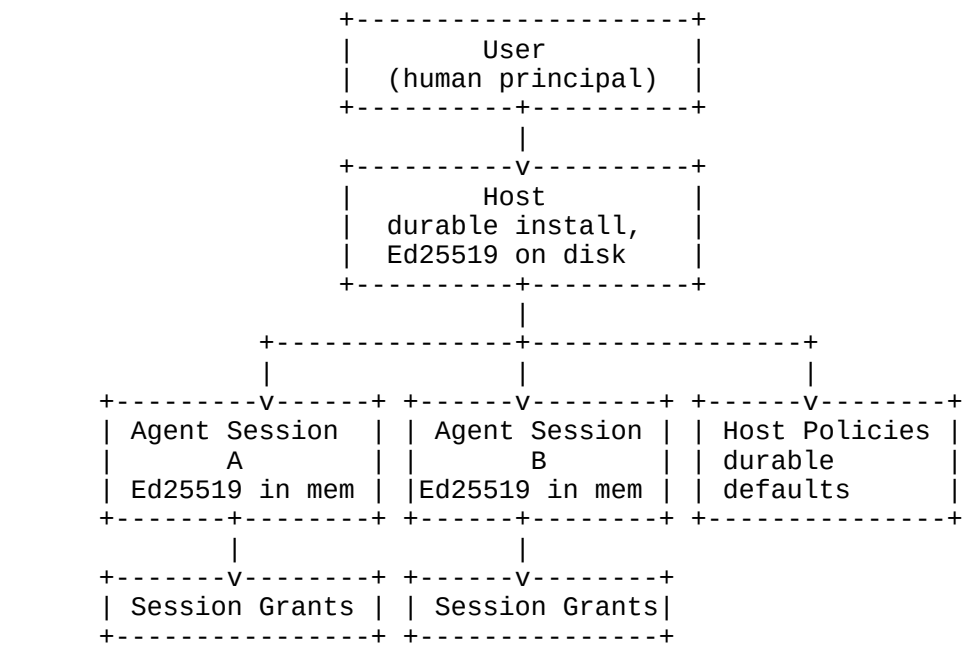
```
{
  "typ": "host-attestation+jwt",
  "alg": "EdDSA"
}
.
{
  "iss": "<hostId>",
  "sub": "agent-registration",
  "iat": 1711000000,
  "exp": 1711000060
}
```

This JWT MUST be signed with the host’s Ed25519 private key and MUST expire within 60 seconds.

Registration sequence:



Identity hierarchy:



4.3. Cross-Party Unlinkability

Agent identifiers in delegated tokens MUST be pairwise per relying party by default. Without pairwise derivation, the session identifier would travel to every relying party as a stable cross-service correlator.

Derivation:

act.sub = MAC(PAIRWISE_SECRET, sector + "." + sessionId)

Where:

* MAC is a keyed message authentication code with at least 128 bits of output. HMAC-SHA-256 meets this requirement.

* PAIRWISE_SECRET is a server-side secret of at least 32 bytes.

* sector is the RP's registered sector identifier, following the same mechanism used for user pairwise sub in VEIL [I-D.valverde-oauth-veil]. Deployments that do not yet support sector_identifier_uri MUST still enforce a stable single-host registration rule so the derived sector remains deterministic.

* sessionId is the internal agent session identifier.

Properties:

* Two RPs receiving tokens from the same agent session see different act.sub values.

* The same derivation applies to agent.id in the PACT delegation claim set.

* Deployments that need global agent tracking MUST use an agent-specific client setting distinct from VEIL's user-facing subject_type. Reusing subject_type would disable pairwise user identifiers at the same time.

* Pairwise derivation uses the session ID (not host ID) because the acting principal is the runtime session, not the installation.

4.4. Trust Gradation

Host attestation assigns a host to one of two trust tiers. Attestation widens the default host policy but does not create a separate token class or silently widen identity-disclosure capabilities.

Tier	How reached	Effect on default host policy
unverified	Default registration	check_compliance, request_approval
attested	Valid OAuth-Client-Attestation + PoP	Same default capability floor; trust tier is surfaced in UI, tokens, and introspection

Table 5

A host's attestation tier is recorded at registration and surfaces in the approval UI (for example, "Verified by <vendor>" versus "Unverified agent"), in token claims (agent.runtime.attested: true/false), and in introspection responses.

5. Named-Action Containment

Capability-based authorization determines whether a session can perform an action without interrupting the user. The registry defines named actions, grants bind them to sessions, constraints restrict parameters, and the usage ledger enforces temporal and cumulative limits.

5.1. The Registry

Capabilities are server-defined named actions. Each capability declares:

Field	Type	Required	Description
name	string	Yes	Stable snake_case identifier
description	string	Yes	Human-readable description
input_schema	JSON Schema	No	Expected input parameters
output_schema	JSON Schema	No	Expected output shape
approval_strength	enum	Yes	none, session, or biometric

Table 6

Discovery. GET {capabilities_endpoint} returns the full registry. No authentication required.

5.2. Grants

A grant links one agent session (or host) to one capability with optional constraints, usage limits, and an expiration time.

Field	Type	Description
capability_name	string	The capability being granted
status	enum	pending, active, denied, revoked
constraints	object	Typed constraint operators (Section 5.3)
daily_limit_count	integer	Max executions per 24-hour window
daily_limit_amount	number	Max cumulative amount per 24-hour window
cooldown_sec	integer	Minimum seconds between executions
source	string	host_policy, session_elevation, or user_grant
expires_at	timestamp	Per-grant TTL, independent of session TTL

Table 7

5.3. Typed Constraints

Constraints restrict the input values a grant authorizes. They use the input schema field names as keys.

Operator	Type	Semantics
max	number	Value MUST be <= max
min	number	Value MUST be >= min
eq	any	Value MUST equal the constraint value
in	array	Value MUST be a member of the array
not_in	array	Value MUST NOT be a member of the array

Table 8

Within one grant, all constraints are AND (all must pass). Across multiple grants for the same capability, the first matching active grant wins (OR).

Constraint extraction. For capability evaluation, the server extracts constraint-matchable values from authorization_details [RFC9396]. The type field maps to a capability name. Nested fields use dot notation (amount.value, amount.currency).

Bidirectional negotiation. The agent proposes constraints at registration, and the server or user can narrow them. The server MUST NOT widen constraints beyond what the agent proposed without a new approval.

Unknown operators. If a grant contains an unrecognized constraint operator, the server MUST reject the evaluation with a constraint_violated error.

5.4. Durable Defaults vs. Ephemeral Elevations

The policy model is split for the same reason identity is split: durable defaults and per-runtime elevations have different lifetimes and different trust properties.

Host policies are durable and host-scoped. They survive across sessions, are seeded by defaults or attestation tier, carry constraints and usage limits, and are modified only by the user or server admin.

Session grants are ephemeral and session-scoped. They belong to one live session, are copied from host policies at session registration (status: active), are created as pending elevations for requested capabilities beyond defaults (status: pending), and expire when the session expires or is revoked.

Seeding sequence at session registration:

1. Ensure the capability registry is populated.

2. Ensure default host policies exist for the host's trust tier.

3. Copy all active host policies into active session grants (source: host_policy).

4. Create pending session grants for requested capabilities not in the defaults (source: session_elevation).

5.5. Temporal and Cumulative Limits

The usage ledger is an append-only table recording each approved execution. It enforces limits on execution frequency and cumulative totals, independent of the per-request constraint checks in Section 5.3.

Scope determination. Usage is scoped to the narrowest applicable boundary:

* If the grant is backed by a host policy: scope to the host policy (shared across sessions).

* If the grant is session-specific: scope to the session grant.

* Fallback: scope to the session.

Enforcement sequence (within a single database transaction):

1. Cooldown check. Query the ledger for any execution within cooldown_sec of now. If found, reject.

2. Daily count check. Count executions in the last 24 hours. If count >= daily_limit_count, reject.

3. Daily amount check. Sum amount in the last 24 hours. If sum + request.amount > daily_limit_amount, reject.

4. Record. Insert a new ledger entry. Return success.

The transaction ensures atomicity; two concurrent requests cannot both pass a limit that only has room for one.

6. Risk-Proportional Consent

At runtime, a delegation request must be routed to one of three outcomes: auto-approved (within pre-authorized limits), interactive approval (user prompt required), or denied. PACT uses CIBA as the transport and defines the routing logic that selects the outcome.

6.1. CIBA as Correlation Point

The CIBA auth_req_id correlates agent identity (from the control plane), human consent (from the approval interaction), token issuance (from the token endpoint), and the audit trail (from the usage ledger). An Agent-Assertion JWT MUST be bound to a specific auth_req_id; it MUST NOT be accepted as general runtime authentication outside the CIBA request it carries.

6.2. Three Routing Outcomes

The consent router produces three outcomes based on the capability's approval_strength and the agent's session grants. They differ along one axis: how much human involvement is required.

Outcome	When	Human interruption
Silent approval	Active grant exists, constraints pass, limits not exceeded, and capability strength is none	None
Session approval	No matching grant, or strength is session	Push notification with inline approve/deny
Biometric approval	Capability strength is biometric	Push notification with "Open to approve" link only; WebAuthn userVerification: required on the approval page

Table 9

What silent approval can never do. The evaluator MUST refuse automatic approval in these cases:

* Any request containing identity scopes (scopes that would release PII).
* Any request whose derived capability is missing from the registry.
* Any capability whose approval strength is biometric.
* Any request without an active matching grant.
* Any request that exceeds cooldown or daily limits.

6.3. Runtime Proof

Before requesting consent, the client signs an Agent-Assertion JWT with the session's Ed25519 private key:

```
{
  "typ": "agent-assertion+jwt",
  "alg": "EdDSA"
}
.
{
  "iss": "<sessionId>",
  "jti": "<unique>",
  "iat": 1711000000,
  "exp": 1711000060,
  "host_id": "<hostId>",
  "task_id": "<unique task identifier>",
  "task_hash": "<SHA-256 hex of binding_message>"
}
```

The assertion is placed in the Agent-Assertion HTTP header on the CIBA backchannel authorize request.

binding_message is the OIDC CIBA Core 1.0 [CIBA-Core] request parameter, defined there as a human-readable display value. PACT retains its human-visible role in the approval UI and additionally requires that a binding_message be present whenever an Agent-Assertion is present. The task_hash claim is defined as SHA-256(binding_message), giving the agent cryptographic commitment to the same string the human will see. An assertion whose task_hash does not match the CIBA request's binding_message MUST NOT produce agent-bound token semantics.

Verification sequence:

1. Decode payload without verification to extract iss (= sessionId).

2. Load session from the database.

3. Verify session status is active.

4. Import session’s stored public key.

5. Verify JWT signature with algorithms: ["EdDSA"].

6. Verify typ header is agent-assertion+jwt.

7. Compute session lifecycle state (Section 9). Reject if not active.

8. Compute SHA-256(binding_message) and compare to task_hash. MUST match.

9. Verify host ownership: the session’s host MUST belong to the same user and OAuth client as the CIBA request.

Binding to CIBA request. On successful verification, the server snapshots server-owned metadata onto the CIBA request record:

Field	Source
agent_session_id	Session record
host_id	Session’s parent host
display_name, runtime, model, version	Session registration metadata
task_id, task_hash	Agent-Assertion claims
assertion_verified	true
pairwise_act_sub	Derived for the requesting OAuth client
attestation_provider, attestation_tier	Host attestation metadata

Table 10

This snapshot ties the later token to the actual registered runtime, not a free-form claim supplied by the client.

CIBA request integrity. Three additional security properties apply to CIBA requests in PACT:

1. Release handle binding. The CIBA release handle (used to exchange an approved request for tokens) MUST be cryptographically bound to (userId, authReqId, clientId). A handle obtained from one request MUST NOT redeem a different request.

2. Atomic status transitions. The transition from approved to redeemed MUST use compare-and-swap semantics. Concurrent polling attempts that race the transition MUST fail rather than duplicate-issue tokens.

3. Entropy. auth_req_id values MUST be generated with cryptographic entropy sufficient to prevent enumeration (e.g., UUID v4).

6.4. Biometric Approval Strength

An agent with browser control (for example, Playwright or an MCP browser tool) can navigate to approval URLs and click buttons. The biometric approval strength defends against self-approval by requiring WebAuthn userVerification: required, a biometric or PIN gesture that the agent cannot produce.

For biometric-strength capabilities:

* Push notifications show "Open to approve" link only; inline approve/deny actions are disabled because they cannot trigger WebAuthn.

* The approval endpoint returns a WebAuthn challenge that MUST be satisfied before the approval is accepted.

* Password-based session re-authentication is insufficient because the agent may know the password.

The session strength accepts any in-session re-authentication an agent with browser access could automate (cookie presence, password re-entry). The biometric strength requires a WebAuthn user-verification ceremony that cannot be performed without the registered authenticator.

Identity scope exception. When a CIBA request includes identity scopes, push notifications MUST NOT include inline approve actions, even for session-strength capabilities. Vault unlock (required by VEIL for identity claim delivery) cannot be triggered from a service worker or agent browser context, making identity-scoped requests functionally equivalent to biometric for inline approval purposes. The user MUST navigate to the approval page in a full browser context.

6.5. Capability Derivation

The consent router derives a capability name from the CIBA request's authorization_details and scope:

Condition	Derived capability
Any detail has type equal to "purchase"	purchase
Any scope is an identity scope	read_profile
Any scope is a proof scope (for example proof:age or proof:compliance)	check_compliance
Only openid scope, no typed details	request_approval

Table 11

The first matching row wins. The precedence order is purchase details, then identity scopes, then proof scopes, then openid-only requests. A mixed request (for example, purchase details plus identity.* scopes) is routed to the most sensitive derived capability.

7. Delegation Evidence

Approved delegations are encoded as JWT access tokens. The sub and act claims identify the user and agent principals. The capabilities, oversight, audit, and delegation claims describe the constraints and provenance.

7.1. The Delegation Token Profile

Access tokens issued after agent-verified CIBA approval carry the delegation claim set defined below. The act claim is used as specified in [RFC8693] Section 4.1. The remaining claims (agent, task, capabilities, oversight, audit, delegation) are defined normatively in this document. Approved authorization_details [RFC9396] are retained on the CIBA request record and projected onto exchanged tokens (Section 8), not on the CIBA-issued access token.

```
{
  "iss": "https://as.example.com",
  "sub": "<pairwise user id for RP>",
  "aud": "<RP client_id>",
  "exp": 1711003600,
  "iat": 1711000000,
  "jti": "<unique>",
  "scope": "openid purchase",

  "act": {
    "sub": "<pairwise agent id for RP>"
  },

  "agent": {
    "id": "<pairwise agent id for RP>",
    "type": "mcp-agent",
    "model": {
      "id": "claude-sonnet-4-20250514",
      "version": "1.0.0"
    },
    "runtime": {
      "environment": "node",
      "attested": true
    }
  },

  "task": {
    "id": "task-uuid",
    "purpose": "purchase"
  },

  "capabilities": [
    {
      "action": "purchase",
      "constraints": [
        { "field": "amount.value", "op": "max", "value": 100 }
      ]
    }
```

```
    ],

    "oversight": {
      "approval_reference": "<auth_req_id>",
      "requires_human_approval_for": ["identity.*"]
    },

    "audit": {
      "trace_id": "<auth_req_id>",
      "session_id": "<pairwise agent id for RP>"
    },

    "cnf": {
      "jkt": "<DPoP key thumbprint>"
    }
  }
```

Claim semantics:

Claim	Semantics
sub	Pairwise user identifier for the target RP
act.sub	Pairwise agent session identifier for the target RP
agent.id	Same as act.sub, pairwise agent identifier
agent.type	Agent runtime type (e.g., mcp-agent)
agent.model	Model metadata (informational, not security-critical)
agent.runtime.attested	Whether the host passed vendor attestation
task.id	Task identifier from the Agent-Assertion
task.purpose	Category-level intent (not verbatim description)
capabilities	Approved capabilities with their constraint snapshot
oversight.requires_human_approval_for	Scopes that MUST route through human approval
audit.trace_id	CIBA auth_req_id for end-to-end correlation
cnf.jkt	DPoP proof-of-possession key thumbprint

Table 12

Conditional emission. If assertion_verified is false on the CIBA request, delegation claims MUST NOT be emitted; the token is issued as a standard CIBA token without agent semantics.

7.2. Chain Tracking

When tokens are exchanged via Token Exchange [RFC8693], the delegation claim tracks the chain:

```
{
  "delegation": {
    "depth": 1,
    "max_depth": 3,
    "chain": ["<pairwise-agent-A>", "<pairwise-agent-B>"],
    "parent_jti": "<original-token-jti>"
  }
}
```

Rules enforced on token exchange:

* The server MUST maintain delegation lineage in canonical internal actor references (for example, raw session IDs), not only in the audience-projected identifiers emitted in tokens.

* depth incremented by 1 on each exchange.

* Current actor appended to the canonical lineage before projection.

* delegation.chain in the exchanged token MUST be projected for the current audience from the canonical lineage. Previous audience-specific pairwise values MUST NOT be copied verbatim into a new audience context.

* Implementations advertising delegation_chains: true MUST reject the exchange if depth >= max_depth.

* Implementations advertising delegation_chains: true MUST enforce mandatory privilege reduction: at least one of narrower capabilities, tighter constraints, shorter TTL, or lower max_depth.

* The first exchange from a CIBA access token to a non-agent audience satisfies privilege reduction by applying the claim-narrowing rules of Section 8: the issued token omits the delegation agent, task, capabilities, oversight, and audit sections, narrows authorization_details to the approved subset, and rebinds audience. If delegation_chains is false, the issued token MAY omit delegation as well; if delegation_chains is true,

it MUST retain the projected delegation lineage needed for depth enforcement and family revocation. The issued token lifetime MUST NOT exceed the subject token's remaining lifetime.

Family revocation. Revoking a parent token (by jti) revokes all descendants reachable via parent_jti graph traversal.

8. Claim Narrowing on Token Exchange

When delegation evidence must cross an audience boundary, the AS uses Token Exchange [RFC8693] to mint a downstream token for the new audience. This section specifies what the AS MUST strip, project, and attenuate on every such exchange. The rules apply regardless of the requested_token_type; PACT does not mint any profile-specific artifact type.

8.1. Dropping Agent Control-Plane Claims

When the target audience is not itself an agent (the common case: a resource server or relying party consuming the delegated authorization), the issued token MUST omit the delegation agent, task, capabilities, oversight, and audit sections. These sections describe the agent control plane and leak agent-identifying context that a non-agent audience does not need for access decisions.

* If the deployment does not advertise delegation_chains, the issued token MAY omit delegation entirely.

* If the deployment advertises delegation_chains: true, the issued token MUST retain a projected delegation claim containing only the lineage needed for depth enforcement and family revocation (depth, max_depth, chain, parent_jti, root_jti; see Section 7). Previous audience-specific pairwise values MUST NOT be copied verbatim into the new audience context and MUST be recomputed per Section 4.3.

Claims that the downstream audience does need (iss, aud, sub, act.sub, scope, authorization_details, cnf, jti, iat, exp) are copied or recomputed from the subject token: pairwise identifiers are recomputed for the target audience (Section 4.3), DPoP binding is taken from the token exchange request's proof, and authorization_details is narrowed per the rules below.

8.2. Privilege Reduction

Every token exchange MUST enforce privilege reduction on the issued token:

* scope MUST be a subset of the subject token's scope.

* authorization_details MUST be a subset of the subject token's authorization_details.

* Lifetime MUST NOT exceed the subject token's remaining lifetime.

8.3. Request and Response

A downstream audience MAY negotiate a specialized requested_token_type (for example, a deployment-specific signed artifact type). The specific types are out of scope for this profile; the rules above apply regardless of the chosen type.

Request:

```
POST {token_endpoint}
  DPoP: <proof bound to the caller key>

  grant_type=urn:ietf:params:oauth:grant-type:token-exchange
  subject_token=<CIBA access token>
  subject_token_type=urn:ietf:params:oauth:token-type:access_token
  requested_token_type=
    urn:ietf:params:oauth:token-type:access_token
  audience=<target RP client_id>
```

Issued token (illustrative):

```
{
  "iss": "https://as.example.com",
  "aud": "<target RP client_id>",
  "sub": "<pairwise user id for target RP>",
  "act": { "sub": "<pairwise agent id for target RP>" },
  "scope": "<narrowed subset>",
  "authorization_details": [
    {
      "type": "purchase",
      "merchant": "Acme",
      "item": "Widget",
      "amount": { "value": "29.99", "currency": "USD" }
    }
  ],
  "cnf": { "jkt": "<DPoP key thumbprint>" },
  "jti": "<unique>",
  "iat": 1711000000,
  "exp": 1711003600
}
```

The target RP validates the JWT signature against the AS's JWKS, verifies a matching DPoP proof against cnf.jkt, and enforces the authorization_details constraints. It does not need to understand the agent capability model; it only needs to trust the signature and proof-of-possession binding.

9. Temporal Boundaries

Agent sessions have bounded lifetimes. Expired or terminated sessions MUST NOT be reactivated.

9.1. Two Independent Clocks

Each agent session has two independent lifetime clocks: one tracks inactivity, the other tracks total elapsed time since session creation.

Clock	Measured from	Default	Purpose
Idle TTL	Last activity (last_seen_at)	1800s (30 min)	Inactivity timeout
Max lifetime	Session creation (created_at)	86400s (24 h)	Total session cap

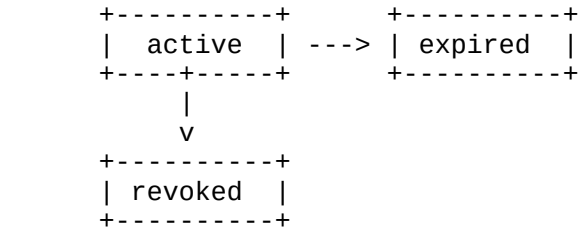
Table 13

Lifecycle computation:

1. If persisted status is revoked or expired: keep it (terminal).
2. Compute idle_expires_at = last_seen_at + idle_ttl_sec * 1000.
3. Compute max_expires_at = created_at + max_lifetime_sec * 1000.
4. If now >= idle_expires_at OR now >= max_expires_at: status is expired.
5. Otherwise: status is active.

Expiry is not only inferred at read time. It MUST be persisted to the database once observed.

9.2. No Reactivation



There is no reactivation path for sessions. If a session expires, the client creates a new session under the same host. This is simpler and more auditable than hidden reactivation logic, and it ensures that escalated session grants do not survive across activation boundaries.

9.3. Renewal Through Use

Session activity (last_seen_at) is updated after successful authenticated operations, specifically after successful Agent-Assertion binding during CIBA requests. The idle boundary is renewed by successful use, not by mere existence.

9.4. Revocation Cascade

An agent session can be revoked by the user (via dashboard or API), the client (via POST {revocation_endpoint} with the bootstrap token carrying agent:session.revoke), or the server (policy-based).

Revoking a session also revokes all its session grants (status = "revoked", revoked_at = now). Revoking a host revokes all sessions under it, cascading to all session grants.

Logout coordination. The authorization server MUST revoke all pending CIBA requests for a user at the time of user logout (per VEIL [I-D.valverde-oauth-veil]). A CIBA token MUST NOT be issuable after the user’s session has been terminated. Without this, an agent polling after user logout could obtain tokens for a session the user intended to end.

10. Machine-Readable Surfaces

PACT exposes its agent authorization model through four machine-readable surfaces: an agent configuration document, a capability registry, token introspection, and an optional A2A agent card.

10.1. Agent Configuration Document

GET /.well-known/agent-configuration is the profile discovery endpoint. No authentication required.

```
{
  "issuer": "https://as.example.com",
  "registration_endpoint":
    "https://as.example.com/api/auth/agent/register",
  "host_registration_endpoint":
    "https://as.example.com/api/auth/agent/host/register",
  "capabilities_endpoint":
    "https://as.example.com/api/auth/agent/capabilities",
  "introspection_endpoint":
    "https://as.example.com/api/auth/agent/introspect",
  "revocation_endpoint":
    "https://as.example.com/api/auth/agent/revoke",
  "jwks_uri": "https://as.example.com/api/auth/agent/jwks",
  "supported_algorithms": ["EdDSA"],
  "approval_methods": ["ciba"],
  "approval_page_url_template":
    "https://as.example.com/approve/{auth_req_id}",
  "supported_features": {
    "task_attestation": true,
    "pairwise_agents": true,
    "risk_graduated_approval": true,
    "capability_constraints": true,
    "delegation_chains": false
  }
}
```

The document SHOULD be served with Cache-Control: public, max-age=3600.

10.2. Capability Discovery

GET {capabilities_endpoint} returns the full capability registry.

GET {capabilities_endpoint}/{name} returns a single capability with full input/output schemas.

No authentication required for either endpoint.

10.3. Introspection with Lifecycle

POST {introspection_endpoint} follows the [RFC7662] model for agent token validation.

Authentication. Requires a client_credentials access token with agent:introspect scope.

Request: application/x-www-form-urlencoded or application/json with a token field.

Key behaviors:

* Session lifecycle is re-evaluated at query time. A session that expired between token issuance and introspection returns active: false.

* sub and agent.id MUST come from a consistent caller-relative pairwise view. If the server cannot safely re-project sub for the introspecting client, it MUST omit sub rather than leak another client's pairwise identifier.

* client_id and aud continue to identify the token that was issued, not the introspector's own client registration.

* If assertion_verified is false on the token snapshot, delegation claims are omitted.

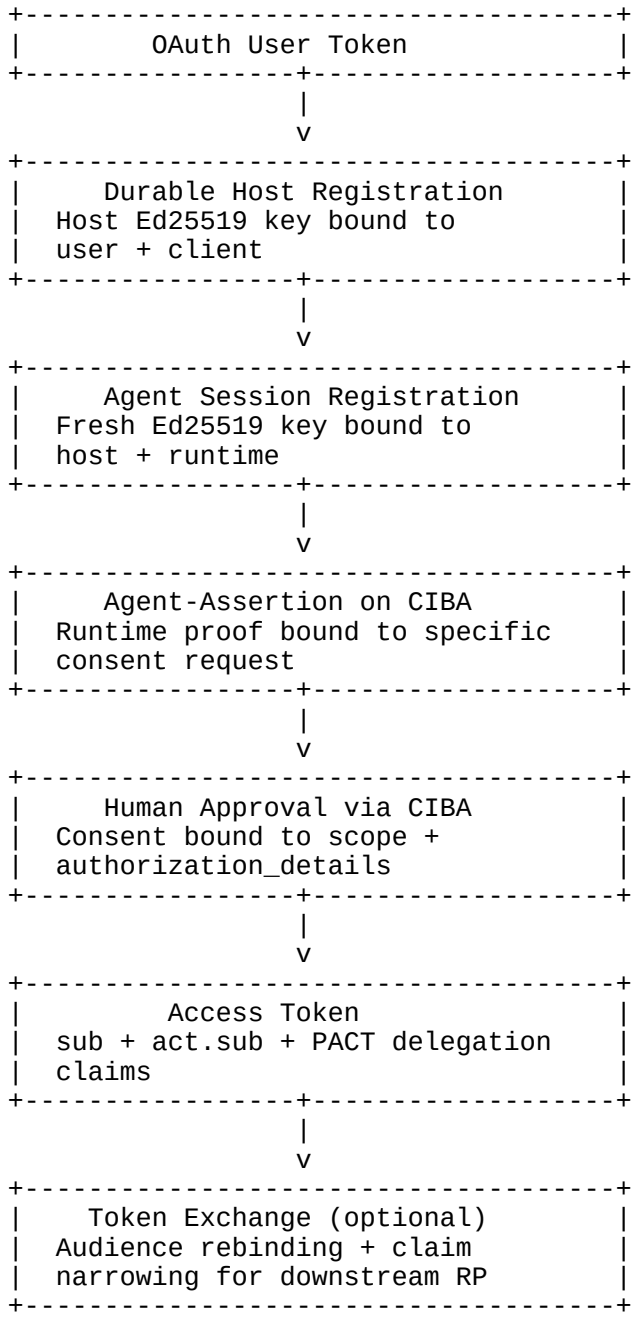
10.4. A2A Agent Card (Informational)

Deployments MAY additionally publish an A2A Protocol [A2A] agent card at GET /.well-known/agent-card.json that references the agent configuration document, to support agent-to-agent capability discovery ecosystems that rely on that surface. This integration is informational: the A2A Protocol is not a normative dependency of this profile, and conforming implementations are not required to emit the A2A card.

11. Cryptographic Continuity

Each phase of the profile produces cryptographic evidence consumed by the next. A relying party verifies the delegation chain by inspecting the resulting OAuth messages; each step requires proof that only the legitimate caller can produce.

11.1. The Full Chain


OAuth User Token
Durable Host Registration
Host Ed25519 key bound to
user + client
Agent Session Registration
Fresh Ed25519 key bound to
host + runtime
Agent-Assertion on CIBA
Runtime proof bound to specific
consent request
Human Approval via CIBA
Consent bound to scope +
authorization_details
Access Token
sub + act.sub + PACT delegation
claims
Token Exchange (optional)
Audience rebinding + claim
narrowing for downstream RP

11.2. Session Binding

The client authenticates with a user-bound OAuth access token, typically sender-constrained through DPoP. This binds delegated setup work to the caller’s proof-of-possession key.

11.3. Host Binding

Host registration binds a durable Ed25519 public key to one user, one OAuth client, and one installation thumbprint. The signed host-attestation+jwt used during session registration proves that the caller still possesses the durable host private key.

11.4. Runtime Binding

Session registration binds a fresh Ed25519 public key to one host, one runtime process, and one display metadata snapshot. The Agent-Assertion proves possession of that runtime key on each CIBA request.

11.5. Consent Binding

CIBA binds human approval to one auth_req_id, one binding message, one scope set, and one optional authorization_details payload.

If the runtime proof is present and valid, the server snapshots runtime metadata into the CIBA request before the approval result is finalized.

11.6. Disclosure Binding

Identity release follows VEIL [I-D.valverde-oauth-veil]: PII is staged in an ephemeral single-consume store and delivered via the userinfo endpoint, never embedded in id_token or access-token claims. PACT sets the CIBA TTL to 10 minutes; the OAuth2 TTL of 5 minutes is inherited unchanged.

11.7. Delegation Binding

The delegated access token carries sub for the human principal, act.sub for the acting agent session, and the full PACT delegation claim set (Section 7). Tokens issued by token exchange to a non-agent audience carry pairwise sub, pairwise act.sub, cnf.jkt, and the approved authorization_details, all re-derived or rebound for the target audience per Section 8.

12. Security Considerations

Implementations MUST satisfy the security considerations of OAuth 2.1 [I-D.ietf-oauth-v2-1], CIBA [CIBA-Core], OAuth Token Exchange [RFC8693], and VEIL [I-D.valverde-oauth-veil].

12.1. JTI Replay Protection

All Agent-Assertion JWTs require a jti claim. The server MUST reject duplicates for the same session and retain seen values until the assertion's exp plus a clock-skew allowance (SHOULD default to 30 seconds). Cache entries are partitioned by session ID.

12.2. Algorithm Confusion Prevention

JWT verification MUST derive the algorithm from the public key's curve (Ed25519 -> EdDSA, P-256 -> ES256), never from the JWT alg header.

12.3. JWKS Fetch Hardening

Remote JWKS URL fetches (for attestation verification) MUST block private and loopback IP addresses, enforce HTTPS in production, limit redirects (max 3), cap response size, enforce timeouts, and cache responses.

12.4. Host Key Security

Host private keys MUST be stored with filesystem permissions restricting access to the owning user only (mode 0600). The directory MUST be mode 0700.

12.5. Session Key Ephemerality

Session private keys MUST exist only in process memory. They MUST NOT be written to disk, environment variables, or any persistent store.

12.6. Task Description Sensitivity

Task descriptions may contain user-specific context. They appear as task.purpose in tokens (category-level, not verbatim). The verbatim description is available only via introspection by authorized machine clients, not in the JWT body.

12.7. DPoP Binding

Delegated access tokens issued by this profile SHOULD be DPoP-bound [RFC9449]. When token exchange mints a downstream token from a CIBA access token, the issued token MUST carry cnf.jkt from the validated DPoP proof and the target RP MUST require a matching DPoP proof at presentation time. A leaked exchanged token is useless without the caller's proof-of-possession key.

13. Privacy Considerations

The privacy considerations of VEIL [I-D.valverde-oauth-veil] apply in addition to those below. Participants are the user, the agent session, the durable host installation, the authorization server, and the relying parties receiving delegated tokens. Cross-RP correlation through agent identifiers requires explicit opt-in at registration.

13.1. Cross-RP Agent Correlation

Without pairwise agent identifiers, act.sub in delegated tokens is a stable correlator across all RPs. Two colluding RPs can link agent activity across services, correlate the frequency and timing of operations, and infer usage patterns that may deanonymize the human.

PACT mitigates this by deriving act.sub per-RP using the same sector-identifier mechanism as OIDC Core pairwise subjects.

13.2. PII in Tokens

Identity PII MUST NOT be embedded in access tokens or purchase authorization artifacts. PII delivery uses the ephemeral in-memory store with single-consume semantics. This ensures that long-lived tokens contain only cryptographic identifiers, not personal data.

13.3. Metadata Minimization

Agent display metadata (model, runtime, version) is informational and self-declared. It SHOULD NOT be relied upon for security decisions. Trust decisions MUST be based on cryptographic verification (key signatures, attestation) rather than declared metadata.

13.4. Pairwise Opt-Out

RPs that require stable agent identifiers (for example, for regulatory audit trails) opt in through an agent-specific signal distinct from VEIL's user-facing subject_type. The default is pairwise.

14. IANA Considerations

This document requests the registrations listed below. Media types under application/*+jwt are registered per [RFC7515].

14.1. Media Type Registrations

This document requests two registrations in the "Media Types" registry under the application/ tree:

* application/agent-assertion+jwt: JWT typ defined in Section 6.3; used in the Agent-Assertion HTTP header to prove runtime session identity on CIBA requests.

* application/host-attestation+jwt: JWT typ defined in Section 4.2; host-signed assertion consumed by the session registration endpoint.

Each registration specifies this document as its reference and references [RFC7515] for the +jwt structured syntax suffix semantics.

14.2. Well-Known URI Registration

This document requests one registration in the "Well-Known URIs" registry [RFC8615]:

* URI suffix: agent-configuration

* Change Controller: IETF

* Reference: This document, Section 10.1

* Status: Permanent

14.3. HTTP Field Name Registrations

This document requests three registrations in the "Hypertext Transfer Protocol (HTTP) Field Name Registry" [RFC9110]:

* Field Name: Agent-Assertion. Status: permanent. Reference: this document Section 6.3.

* Field Name: OAuth-Client-Attestation. Status: permanent. Reference: [I-D.ietf-oauth-attestation-based-client-auth].

* Field Name: OAuth-Client-Attestation-PoP. Status: permanent. Reference: [I-D.ietf-oauth-attestation-based-client-auth].

The latter two fields are registered by their originating draft; PACT references them without redefinition.

14.4. JWT Claim Registrations

This document requests registrations in the "JSON Web Token Claims" registry [RFC7519] for the delegation claim vocabulary defined in Section 7:

* agent: Object describing the acting agent runtime and attestation posture.

* task: Object describing the task category under which delegation was authorized.

* capabilities: Array of approved named actions with typed constraint snapshots.

* oversight: Object carrying human-approval requirements and the originating approval reference.

* audit: Object carrying a CIBA trace identifier and session reference.

* delegation: Object tracking the delegation chain.

The act, authorization_details, cnf, jti, iat, exp, iss, aud, sub, and scope claims are referenced as already registered and are used per their originating specifications ([RFC8693] Section 4.1 for act, [RFC9396] for authorization_details, [RFC7800] for cnf, [RFC7519] for the common set).

14.5. OAuth Extensions Error Registry

PACT does not currently request new OAuth error codes; it uses the step-up error contract inherited from VEIL (interaction_required) and standard CIBA and token-exchange errors defined in their respective specifications.

14.6. ACR Values

ACR URNs in acr_values_supported are deployment-local (see VEIL [I-D.valverde-oauth-veil]). PACT does not register ACR values or reserve an ACR namespace.

15. Conformance

15.1. Server Requirements

A conforming authorization server MUST:

* Implement the identity model.
* Compute pairwise agent identifiers by default.
* Implement the capability registry and grant model.
* Route consent based on approval strength.
* Verify Agent-Assertions and bind them to CIBA requests.
* Issue tokens with PACT delegation claims only when assertions are verified.
* Publish the agent configuration document.
* Enforce JTI replay protection.
* Derive JWT algorithms from public keys, not headers.

A conforming authorization server SHOULD:

* Support vendor attestation.
* Enforce usage limits atomically.
* Support token exchange with claim narrowing for downstream audiences.
* Provide introspection with lifecycle evaluation.
* Bind tokens with DPoP.

15.2. Client Requirements

A conforming client MUST:

* Generate and persist Ed25519 host keys.
* Generate ephemeral Ed25519 session keys in memory only.
* Sign host-attestation+jwt for session registration.

* Sign Agent-Assertion JWTs before CIBA requests.

* Include jti in all signed JWTs.

A conforming client SHOULD:

* Present vendor attestation headers when available.

* Request only the capabilities it needs.

* Propose constraints that reflect its actual intended usage.

16. References

16.1. Normative References

[CIBA-Core]
OpenID Foundation, "OpenID Connect Client-Initiated Backchannel Authentication Core 1.0", n.d., <https://openid.net/specs/openid-connect-client-initiated-backchannel-authentication-core-1_0.html>.

[I-D.ietf-oauth-attestation-based-client-auth]
"OAuth 2.0 Attestation-Based Client Authentication", n.d., <https://datatracker.ietf.org/doc/draft-ietf-oauth-attestation-based-client-auth/>.

[I-D.ietf-oauth-v2-1]
Hardt, D., Parecki, A., and T. Lodderstedt, "The OAuth 2.1 Authorization Framework", Work in Progress, Internet-Draft, draft-ietf-oauth-v2-1-15, 2 March 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-oauth-v2-1-15>.

[OIDC-Core]
OpenID Foundation, "OpenID Connect Core 1.0 incorporating errata set 2", 15 December 2023, <https://openid.net/specs/openid-connect-core-1_0.html>.

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

[RFC7515] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", RFC 7515, DOI 10.17487/RFC7515, May 2015, <https://www.rfc-editor.org/rfc/rfc7515>.

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

[RFC7636] Sakimura, N., Ed., Bradley, J., and N. Agarwal, "Proof Key for Code Exchange by OAuth Public Clients", RFC 7636, DOI 10.17487/RFC7636, September 2015, <https://www.rfc-editor.org/rfc/rfc7636>.

[RFC7638] Jones, M. and N. Sakimura, "JSON Web Key (JWK) Thumbprint", RFC 7638, DOI 10.17487/RFC7638, September 2015, <https://www.rfc-editor.org/rfc/rfc7638>.

[RFC7662] Richer, J., Ed., "OAuth 2.0 Token Introspection", RFC 7662, DOI 10.17487/RFC7662, October 2015, <https://www.rfc-editor.org/rfc/rfc7662>.

[RFC7800] Jones, M., Bradley, J., and H. Tschofenig, "Proof-of-Possession Key Semantics for JSON Web Tokens (JWTs)", RFC 7800, DOI 10.17487/RFC7800, April 2016, <https://www.rfc-editor.org/rfc/rfc7800>.

[RFC8037] Liusvaara, I., "CFRG Elliptic Curve Diffie-Hellman (ECDH) and Signatures in JSON Object Signing and Encryption (JOSE)", RFC 8037, DOI 10.17487/RFC8037, January 2017, <https://www.rfc-editor.org/rfc/rfc8037>.

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

[RFC8615] Nottingham, M., "Well-Known Uniform Resource Identifiers (URIs)", RFC 8615, DOI 10.17487/RFC8615, May 2019, <https://www.rfc-editor.org/rfc/rfc8615>.

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

[RFC9110] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Semantics", STD 97, RFC 9110, DOI 10.17487/RFC9110, June 2022, <https://www.rfc-editor.org/rfc/rfc9110>.

[RFC9126] Lodderstedt, T., Campbell, B., Sakimura, N., Tonge, D., and F. Skokan, "OAuth 2.0 Pushed Authorization Requests", RFC 9126, DOI 10.17487/RFC9126, September 2021, <https://www.rfc-editor.org/rfc/rfc9126>.

[RFC9396] Lodderstedt, T., Richer, J., and B. Campbell, "OAuth 2.0 Rich Authorization Requests", RFC 9396, DOI 10.17487/RFC9396, May 2023, <https://www.rfc-editor.org/rfc/rfc9396>.

[RFC9449] Fett, D., Campbell, B., Bradley, J., Lodderstedt, T., Jones, M., and D. Waite, "OAuth 2.0 Demonstrating Proof of Possession (DPoP)", RFC 9449, DOI 10.17487/RFC9449, September 2023, <https://www.rfc-editor.org/rfc/rfc9449>.

16.2. Informative References

[A2A] "Agent-to-Agent Protocol", n.d., <https://a2a-protocol.org>.

[AAP] "Agent Auth Protocol v1.0-draft", n.d., <https://agent-auth-protocol.com/specification/v1.0-draft>.

[I-D.aap-oauth-profile] "Agent Authorization Profile for OAuth", n.d., <https://datatracker.ietf.org/doc/draft-aap-oauth-profile/>.

[I-D.oauth-ai-agents-on-behalf-of-user] "AI Agents Acting on Behalf of Users", n.d., <https://datatracker.ietf.org/doc/draft-oauth-ai-agents-on-behalf-of-user/>.

[I-D.valverde-oauth-veil] Valverde, G., "VEIL: Verified Ephemeral Identity Layer for OAuth 2.1", 2026.

Acknowledgments

The author thanks the OAuth working group and the OpenID Foundation CIBA editors for ongoing work that this profile composes with, and the authors of related individual submissions cited in the Relation to Other Work section for claim-name alignment and shared concepts.

Author’s Address

Gustavo Valverde
Zentity
Portugal
Email: g.valverde02@gmail.com