Agent Control Protocol

ACP v1.15: Admission Control for Agent Actions

Technical Specification and Reference Implementation

Marcelo Fernandez
TraslaIA
info@traslaia.com
https://agentcontrolprotocol.xyz

March 2026
Draft Standard
arXiv:2603.18829 [cs.AI] DOI: 10.5281/zenodo.19150239

Abstract

Agent Control Protocol (ACP) is a formal technical specification for governance of autonomous agents in B2B institutional environments. ACP is the **admission control layer between agent intent and system state mutation**: before any agent action reaches execution, it must pass a cryptographic admission check that validates identity, capability scope, delegation chain, and policy compliance simultaneously.

ACP defines the mechanisms of cryptographic identity, capability-based authorization, deterministic risk evaluation, verifiable chained delegation, transitive revocation, and immutable auditing that a system must implement for autonomous agents to operate under explicit institutional control.

ACP operates as an additional layer on top of RBAC and Zero Trust, without replacing them. It is designed specifically for the problem that neither model solves: governing what an autonomous agent can do, under what conditions, with what limits, and with complete traceability for external auditing—including across organizational boundaries.

The v1.15 specification comprises 36 technical documents organized into five conformance levels (L1–L5). It includes a Go reference implementation of 22 packages covering all L1–L4 capabilities, 73 signed conformance test vectors (Ed25519 + SHA-256), an OpenAPI 3.1.0 specification (17 endpoints), and an executable multi-organization interoperability demo. It defines more than 62 verifiable requirements, 12 prohibited behaviors, and the mechanisms for interoperability between institutions.

Specification and implementation: https://github.com/chelof100/acp-framework-en

Contents

1 The Problem ACP Solves

Autonomous agents are being deployed in institutional environments without a technical standard to govern their behavior. This is not a tooling problem—it is a protocol problem.

1.1 The Structural Gap

When an autonomous agent makes a decision and executes it, there is a critical moment between both actions. In current models, that moment does not formally exist: the decision and the execution are the same event. The agent decides and acts. There is no intermediate validation. There is no point of intervention. There is no structured record of why the decision was made.

This is acceptable when a human executes that action, because the human bears responsibility and can be questioned. An autonomous agent cannot be questioned. It can only be audited—and only if there is something to audit.

> *The problem is not whether agents are trustworthy. The problem is that currently no formal technical mechanism exists that allows an institution to demonstrate that its agents operated within authorized limits.*

1.2 ACP as Admission Control

The clearest frame for understanding what ACP does is the Kubernetes Admission Controller analogy.

Kubernetes intercepts every API request before it reaches the cluster and runs it through a sequence of admission checks—`ValidatingWebhookConfiguration`, `ResourceQuota` enforcement, OPA Gatekeeper policies. If any check fails, the request is rejected before touching cluster state.

ACP applies this pattern to agent actions:

```
agent intent
    |
[1] Identity check     (ACP-AGENT-1.0, ACP-HP-1.0)
    |    is this agent who they claim to be?
[2] Capability check   (ACP-CT-1.0, ACP-DCMA-1.1)
    |    does the agent hold a valid token for this action?
[3] Policy check       (ACP-RISK-1.0, ACP-PSN-1.0)
    |    is this action within current policy?
[4] ADMIT / DENY / ESCALATE
    |    (if ADMIT)
[5] Execution token    (ACP-EXEC-1.0)
    |    single-use cryptographic proof of admission
[6] Ledger record      (ACP-LEDGER-1.3)
    |    immutable signed audit entry
system state mutation
```

The critical difference from Kubernetes: ACP's admission check operates across institutional boundaries. An agent from Bank A can be admitted by Bank B without Bank B trusting Bank A's

internal infrastructure—the cryptographic proof is self-contained and verifiable with Bank A's published public key alone.

This *admission control* framing also clarifies the relationship with related tools:

- **OPA (Open Policy Agent)** [3] can serve as the policy evaluation engine inside Step 3. ACP does not replace OPA; it adds the identity and delegation chain layers above it.
- **AWS IAM / Azure RBAC** model static role permissions for humans. ACP adds dynamic agent delegation with execution proof.
- **OAuth 2.0** [1] handles API access tokens. ACP extends delegation to multi-agent chains with non-escalation and verifiable provenance.
- **SPIFFE / SPIRE** [5] provides cryptographic workload identity. ACP builds on that identity to add capability scoping and governance.

1.3 Why RBAC and Zero Trust Are Insufficient

RBAC and Zero Trust are the predominant control layers in enterprise environments. Both are necessary. Neither solves the problem of governing autonomous agents:

Criterion	RBAC	Zero Trust	ACP
Designed for	Human roles	Network access	Autonomous agents
Cryptographic identity	No	Partial	Yes (Ed25519, mandatory)
Verifiable dynamic delegation	No	No	Yes (chained, auditable)
Decision/execution separation	No	No	Yes (Execution Tokens)
Real-time risk evaluation	No	Partial	Yes (deterministic)
Multi-institutional auditing	Non-standard	Non-standard	Native (signed ledger)
Transitive delegation revocation	No	No	Yes (formal propagation)
B2B interoperability for agents	Unstructured	Unstructured	Central protocol design

Table 1: Comparison of ACP with RBAC and Zero Trust across agent governance criteria.

ACP does not replace RBAC or Zero Trust. It adds a governance layer oriented specifically to autonomous agents that operates above existing controls.

1.4 The Concrete Scenario ACP Prevents

Without ACP: A financial processing agent receives instructions from another agent to execute a transfer. The agent executes the action. If the instruction was legitimate, everything works. If it was compromised, injected, or generated by an unauthorized agent, the transfer occurs anyway. No formal mechanism exists to prevent it, nor is there a technical record that allows reconstructing the authorization chain.

With ACP: The agent requesting the transfer must present a Capability Token cryptographically signed by the issuing institution, demonstrate possession of the associated private key, and the request passes through the risk engine before receiving an Execution Token. The ET is single-use. The entire chain is recorded in the Audit Ledger with an externally verifiable institutional signature.

2 What ACP Is

A formal technical specification—not a framework, not a platform, not a set of best practices. A protocol with precise definitions, formal state models, verifiable flows, and explicit conformance requirements.

2.1 Definition

Agent Control Protocol (ACP) is a technical specification that defines the mechanisms by which autonomous institutional agents are identified, authorized, monitored, and governed in B2B environments. It establishes the formal contract between an agent, the institution that operates it, and the institutions with which it interacts.

Core invariant:

$$\textit{Execute}(\text{request}) \implies \textit{ValidIdentity}(\text{agent}) \land \textit{ValidCapability} \land \textit{ValidDelegationChain} \land \textit{AcceptableRisk}$$

No action of an ACP agent can be executed without all four predicates being simultaneously true. If any fails, the action is denied. No exceptions.

2.2 Design Principles

ACP was designed with five principles that are non-negotiable at implementation time:

- **P1 Fail Closed.** On any internal component failure, the action is denied. Never approved by default.
- **P2 Identity is cryptography.** AgentID = base58(SHA-256(public_key)). No usernames. No arbitrary IDs. Identity cannot be claimed—it must be demonstrated in every request.
- **P3 Delegation does not expand privileges.** The delegated agent's permissions are always a strict subset of the delegator's permissions. This property is cryptographically verified at every chain hop.
- **P4 Complete auditability.** Every decision—approved, denied, or escalated—is recorded in an append-only ledger with institutional signature. Not just successes. Everything.
- **P5 External verification possible.** Any institution can verify ACP artifacts from another institution using only the public key registered in the ITA. No dependency on proprietary systems.

2.3 Formal Agent Model

In ACP, an agent is a formal tuple with well-defined state:

$$A = (\,\text{AgentID},\ \text{capabilities},\ \text{autonomy\_level},\ \text{state},\ \text{limits}\,)$$

Field	Type	Description
AgentID	String (43–44 chars)	`base58(SHA-256(pk))`. Derived from public key. Immutable.
capabilities	List of strings	Explicit permissions. Format: `acp:cap:<domain>.<action>`. Never abstract roles.
autonomy_level	Integer 0–4	Determines risk evaluation thresholds. 0 = no autonomy. 4 = maximum.
state	Enum	`active | restricted | suspended | revoked`. Transition to `revoked` is unidirectional.
limits	Object	Rate limits, maximum amounts, time windows. Not modifiable at runtime.

Table 2: ACP formal agent tuple fields.

2.4 Layered Architecture

ACP does not replace existing security infrastructure. It is added as an upper layer:

```
ACP Layer   -- Autonomous agent governance: identity, authorization, risk, auditing
RBAC Layer  -- Role-based access control for human users
Zero Trust  -- Continuous identity and network access verification
```

3 Technical Mechanisms

ACP defines six interdependent mechanisms. Each has its own formal specification, state model, data structure, protocol flow, and error codes.

3.1 Serialization and Signing (ACP-SIGN-1.0)

Every verification in ACP begins with signature verification. ACP-SIGN-1.0 defines the exact process that produces a binary result—valid or invalid—without ambiguity:

1. **Canonicalization with JCS (RFC 8785) [4].** Produces a deterministic representation of the JSON object, independent of field order and the system that generated it.
2. **SHA-256 hash** over the canonical output in UTF-8.
3. **Ed25519 signature (RFC 8032)** [2] over the hash. 32-byte key, 64-byte signature.
4. **Base64url encoding** without padding for transmission.

Signature verification precedes all semantic validation. An object with an invalid signature is rejected without processing its content. This rule has no exceptions (PROHIB-003, PROHIB-012).

3.2 Capability Token (ACP-CT-1.0)

The Capability Token is ACP's central artifact. It is a signed JSON object that specifies exactly what an agent can do, on what resource, for how long, and whether it can delegate that capability to other agents.

```
{
  "ver": "1.0",
  "iss": "<AgentID_issuer>",
  "sub": "<AgentID_subject>",
  "cap": ["acp:cap:financial.payment"],
  "res": "org.example/accounts/ACC-001",
  "exp": 1718923600,
  "nonce": "<128bit_CSPRNG_base64url>",
  "deleg": { "allowed": true, "max_depth": 2 },
  "parent_hash": null,
  "sig": "<Ed25519_base64url>"
}
```

Critical fields: `exp` is mandatory—a token without expiry is invalid by definition. The 128-bit nonce prevents replay attacks. `parent_hash` chains delegated tokens in a verifiable way. The signature covers all fields except `sig`.

3.3 Handshake and Proof-of-Possession (ACP-HP-1.0)

Possessing a valid Capability Token is not sufficient to act. ACP-HP-1.0 requires that the bearer demonstrate in every request that they possess the private key corresponding to the AgentID declared in the token. This eliminates the possibility of impersonating an agent with a stolen token.

The protocol is stateless—it does not establish sessions, does not produce a `session_id`, does not require server-side state between requests. The proof occurs in every interaction:

1. The receiving system issues a 128-bit challenge generated by CSPRNG, valid for 30 seconds and single-use.
2. The agent signs the challenge together with the HTTP method, path, and body hash of the request.
3. The receiver verifies the signature using the agent's public key, obtained from the ITA.
4. The challenge is deleted immediately after use—it cannot be reused.

This sequence guarantees four formal properties: identity authentication, cryptographic request binding, anti-replay, and transport channel independence.

3.4 Deterministic Risk Evaluation (ACP-RISK-1.0)

Each authorization request passes through a deterministic risk function that produces a Risk Score (RS) in the range $[0, 100]$. The same input always produces the same result—no stochastic elements, no machine learning in the critical path.

$$\text{RS} = \min\big(100,\ B(c) + F_{\text{ctx}}(x) + F_{\text{hist}}(h) + F_{\text{res}}(r)\big)$$

Factor	Description	Example values
$B(c)$	Baseline by capability	`*.read = 0`, `financial.payment = 35`
$F_{\text{ctx}}(x)$	Request context	Non-corporate IP +20; outside hours +15
$F_{\text{hist}}(h)$	Agent history (24h)	Recent denial +20; anomalous frequency +15
$F_{\text{res}}(r)$	Resource classification	`public = 0`; `sensitive = 15`; `restricted = 45`

Table 3: ACP-RISK-1.0 risk scoring factors.

The RS determines the decision according to the thresholds configured for the agent's `autonomy_level`. With `autonomy_level` 2 (standard): RS $\leq 39 \rightarrow$ APPROVED; RS $\in [40, 69] \rightarrow$ ESCALATED; RS $\geq 70 \rightarrow$ DENIED. An agent with `autonomy_level` 0 always receives DENIED.

3.5 Verifiable Chained Delegation (ACP-DCMA-1.1)

ACP allows an agent to delegate capabilities to another agent, which in turn can delegate to a third, up to the maximum depth defined in the root token. Delegation guarantees three properties:

- **No privilege escalation.** The delegated agent's capability set is always $\subseteq$ the delegator's set. Cryptographically verified at every hop via the `parent_hash` field.
- **Bounded depth.** The `max_depth` field establishes the chain limit. A chain that exceeds it is invalid.
- **Transitive revocation.** Revoking an agent's token automatically invalidates all delegated tokens that descend from it.

3.6 Execution Token (ACP-EXEC-1.0)

The separation between authorization and execution is a core principle of ACP. When the authorization engine approves a request, it returns an Execution Token (ET): a single-use artifact with a short lifetime that authorizes exactly that action, on that resource, at that moment.

- An ET can only be consumed once. A second presentation is rejected (PROHIB-002).
- An expired ET is invalid even if it was never used.
- The target system that consumes the ET notifies the ACP endpoint, closing the audit cycle.

3.7 Audit Ledger (ACP-LEDGER-1.3)

The Audit Ledger is a chain of cryptographically signed events where each event includes the hash of the previous event:

$$h_n = \text{SHA-256}(\, e_n \parallel h_{n-1} \,)$$

This structure makes it impossible to modify or delete an event without invalidating the entire subsequent chain. The ledger records all lifecycle event types: GENESIS, AUTHORIZATION (including DENIED and ESCALATED), RISK_EVALUATION, TOKEN_ISSUED, TOKEN_REVOKED, EXECUTION_TOKEN_ISSUED, and EXECUTION_TOKEN_CONSUMED.

Institutions with FULL conformance expose the ledger via `GET /acp/v1/audit/query`, allowing external partners to verify chain integrity using only the institutional ITA public key. Modifying or deleting ledger events is prohibited (PROHIB-007, PROHIB-008).

It is important to note that the ledger provides **verifiable evidence** of execution, not enforcement itself. Enforcement in ACP occurs at the policy evaluation and execution layers (ACP-EXEC-1.0, ACP-DCMA-1.1): the Execution Token is the cryptographic artifact that gates actual system-state mutation. The ledger's role is to make every admission decision—and its outcome—auditable and tamper-evident after the fact.

4 Inter-Institutional Trust

In a B2B environment, agents of one institution interact with another institution's systems. ACP defines the exact mechanism by which this trust is established, verified, and can be revoked.

4.1 Institutional Trust Anchor (ACP-ITA-1.0)

The ITA is the authoritative registry that links an `institution_id` to an Ed25519 public key. It is the only point where ACP depends on an out-of-band mechanism: the initial distribution of the ITA authority's public key. Once that key is resolved, all subsequent verification is autonomous and cryptographic.

Each institution registers in the ITA its Root Institutional Key (RIK)—the private key held in HSM that never leaves it. All ACP artifacts from that institution (tokens, ledger events, API responses) are signed with that key. Any third party can verify them by resolving the public key from the ITA.

4.2 Mutual Recognition (ACP-ITA-1.1)

When two institutions operate under different ITA authorities, ACP-ITA-1.1 defines the mutual recognition protocol. The process requires both authorities to sign a Mutual Recognition Agreement (MRA) establishing: the scope of recognition, the agreement's validity period, and the proxy resolution mechanism.

Recognition is explicitly non-transitive. If A recognizes B and B recognizes C, A does not automatically recognize C. Each bilateral relationship requires its own signed MRA. This prevents uncontrolled expansion of the trust graph.

4.3 Institutional Key Rotation and Revocation

Normal rotation includes a transition period of up to 7 days during which both keys are valid, allowing artifacts signed with the old key to be verified during the transition.

Emergency rotation is activated when a key is compromised. The result is immediate: the key is marked `revoked`, all artifacts signed with it are invalid from that moment, and there is no transition period.

5 Cross-Organization Execution

ACP-CROSS-ORG-1.1 defines a fault-tolerant bilateral protocol for interactions between independent institutions, each operating its own ACP ledger. The protocol closes five gaps identified in the 1.0 version: undefined status tracking, absent retry protocol, unregistered ACK event type, `pending_review` without SLA, and a stale header reference.

5.1 Interaction Model

A cross-organization interaction involves two ACP-compliant institutions: a **source institution** that originates the bundle and a **target institution** that validates, executes, and acknowledges it. The protocol operates asynchronously—the source does not block waiting for an ACK.

Each interaction carries a mandatory `interaction_id` (UUIDv7), immutable and reused across all retries. This is distinct from `event_id` (UUIDv4), which is unique per emission. The target deduplicates by `interaction_id`, not by `event_id`.

For example, an agent in Institution A may request execution in Institution B. The request is recorded as a `CROSS_ORG_INTERACTION` event in A's ledger, transmitted to B, validated against B's policy and capability registry, and resolved through a `CROSS_ORG_ACK` event registered in both ledgers. Institution A derives final execution state from the ACK without storing mutable status.

5.2 Fault Tolerance and Retry Protocol

When no `CROSS_ORG_ACK` is received within the 300-second timeout, the source retries up to three times with exponential backoff:

Attempt	Wait after timeout
1 (initial)	—
2	+30 s
3	+60 s
4 (final)	+120 s

After three failed attempts the interaction transitions to `retry_exhausted` (CROSS-012) and an operational alert is raised. If a valid ACK arrives at any point, all retry timers are cancelled immediately.

5.3 Derived Interaction Status

The status of a cross-organization interaction is never stored as mutable state. It is always derived from events present in the ledger:

Derived status	Condition
`pending_ack`	`CROSS_ORG_INTERACTION` exists, no ACK
`acked`	ACK with `status = accepted`
`rejected`	ACK with `status = rejected`
`pending_review`	ACK with `status = pending_review`
`expired`	`pending_review` AND `now > review_deadline`

Precedence rule: `accepted` > `rejected` > `pending_review`. A mutable `CROSS_ORG_STATUS_UPDATE` event is explicitly prohibited as an anti-pattern that introduces race conditions and divergent audit trails.

5.4 Interaction State Invariants

For any `interaction_id`, ACP enforces the following invariants:

- **At most one terminal state.** Only one `accepted` or `rejected` ACK may exist per `interaction_id`; a duplicate with a different payload is rejected (CROSS-014).
- **ACK dominates retries.** A valid `CROSS_ORG_ACK` cancels all pending retry timers immediately, regardless of attempt number.
- **Immutable correlation key.** Retry operations MUST NOT create new `interaction_id` values; only `event_id` changes across attempts.
- **No terminal-to-non-terminal regression.** Once `accepted` or `rejected`, the status cannot change. The transition `pending_review` → `pending_review` is also prohibited (CROSS-015).

5.5 Pending Review SLA

When the target returns `pending_review`, a 24-hour SLA applies. The ACK payload includes a `review_deadline` field (Unix seconds). Valid terminal transitions are: `accepted`, `rejected`, or `expired` (implicit, when `now > review_deadline`).

5.6 CROSS_ORG_ACK as a First-Class Ledger Event

`CROSS_ORG_ACK` is registered in ACP-LEDGER-1.3 §5.15 as a first-class event type, signed with Ed25519 and serialized via JCS (RFC 8785). This enables verifiable and auditable execution across independent institutional boundaries, eliminating the audit gap that existed in version 1.0.

5.7 Security Considerations

ACP-CROSS-ORG-1.1 assumes authenticated communication channels between institutions. All `CROSS_ORG_INTERACTION` and `CROSS_ORG_ACK` events are signed with the originating institution's

Ed25519 ITA key and serialized via JCS (RFC 8785), providing authenticity, integrity, and non-repudiation.

Without this cryptographic layer, cross-organization events would be subject to forgery, replay, and unauthorized status derivation. Specifically, ACP mitigates:

- **Replay attacks:** `interaction_id` (UUIDv7) and `event_id` (UUIDv4) allow the target to detect and reject duplicate submissions.
- **Event forgery:** Ed25519 signatures over JCS-canonical payloads prevent an adversary from injecting valid-looking ACK or interaction events.
- **Audit divergence:** Because `CROSS_ORG_ACK` is a first-class ledger event on both sides, independent auditors can verify consistency without trusting either institution's mutable state.

ACP does not eliminate all sources of failure in distributed systems, but provides a structured model for detecting, handling, and auditing them.

6 Reputation Snapshot Portability (ACP-REP-PORTABILITY-1.1)

ACP-REP-PORTABILITY-1.1 defines the `ReputationSnapshot`: a compact, cryptographically signed record that carries an agent's reputation score across organizational boundaries. Unlike the bilateral attestation protocol of v1.0, this specification focuses on the snapshot object itself—its structure, signing procedure, validation algorithm, and expiration semantics—enabling any verifier to independently validate a snapshot without trusting an intermediary.

6.1 ReputationSnapshot Structure

A `ReputationSnapshot` contains ten fields: `ver` (version), `rep_id` (UUID v4), `subject_id`, `issuer`, `score` (float), `scale` (`"0-1"` or `"0-100"`), `model_id`, `evaluated_at` (Unix seconds), `valid_until` (Unix seconds), and `signature` (Ed25519, base64url). Fields `scale`, `model_id`, and `valid_until` are new in v1.1; v1.0 snapshots omit them and are accepted without expiration enforcement (backward compatibility, §12).

6.2 Signing Procedure

The signature covers all fields except `signature` itself, serialized via JCS (RFC 8785) canonicalization, SHA-256 digest, and Ed25519 signing:

```
1. raw = json.Marshal(signableReputation)
2. canonical = jcs.Transform(raw)                    // RFC 8785
3. digest = SHA-256(canonical)
4. sig = Ed25519.Sign(privKey, digest)
5. snapshot.signature = base64url(sig)               // no padding
```

JCS canonicalization is mandatory. Using `json.Marshal` directly is prohibited: key ordering differs across Go, Python, and TypeScript implementations, producing verification failures in cross-org deployments.

6.3 Validation Invariants

The validator enforces five invariants (REP-001 through REP-011): (1) `evaluated_at` $\leq$ `valid_until` prevents retroactively backdated snapshots; (2) `now` $\leq$ `valid_until` enforces expiration (v1.1 only, error REP-011); (3) score within `scale` bounds (REP-002); (4) non-empty `issuer` (REP-004); (5) valid Ed25519 signature (REP-010).

Structural validation (`Validate`) and cryptographic verification (`VerifySig`) are intentionally separate operations, enabling lightweight ingestion-time checks without requiring the issuer's public key.

6.4 Divergence Semantics

When a verifier receives snapshots of the same `subject_id` from multiple issuers, it may compute divergence: $|a.\text{score} - b.\text{score}|$. If the divergence exceeds a configurable threshold (default 0.30 for `scale="0-1"`), the verifier emits warning REP-WARN-002. This is non-blocking—the policy decision of whether to accept, escalate, or reject remains with the verifier's business logic, preserving institutional sovereignty.

7 Multi-Organization Interoperability Demo (GAP-14)

The `examples/multi-org-demo/` directory provides an executable end-to-end demonstration of cross-organizational ACP exchange, combining ACP-POLICY-CTX-1.1 and ACP-REP-PORTABILITY-1.1 in a single runnable scenario deployable with Docker in under five minutes.

7.1 Scenario

Two independent organizations, Org-A and Org-B, interact over HTTP. Org-A is the *issuer*: it evaluates an agent request against its local policy and produces two signed artifacts—a `PolicyContextSnapshot` and a `ReputationSnapshot`—using its Ed25519 institutional key. Org-B is the *verifier*: it independently validates both artifacts using the `pkg/policyctx` and `pkg/reputation` packages (no logic duplicated) and renders an autonomous admission decision.

The validation sequence in Org-B is:

1. `policyctx.VerifySig(pcs, pubKey)` — verifies institutional signature (ACP-POLICY-CTX-1.1 §6 step 12).
2. `policyctx.VerifyCaptureFreshness(pcs, 300s)` — enforces $\delta_{\max}$ freshness invariant.
3. `reputation.Validate(rep, now)` — structural invariants REP-001 through REP-011.
4. `reputation.VerifySig(rep, pubKey)` — Ed25519/JCS/SHA-256 cryptographic verification.
5. `reputation.CheckDivergence(orgA, orgB, 0.30)` — divergence check against Org-B's own score.
6. Final ACCEPT / DENY decision under Org-B's sovereign policy.

7.2 Divergence Handling and Institutional Sovereignty

If Org-B holds its own score for the same `subject_id`, it computes divergence $|a.\text{score} - b.\text{score}|$. If this exceeds 0.30, Org-B emits REP-WARN-002. REP-WARN-002 is non-blocking—the final decision is Org-B's prerogative, independent of Org-A's assessment.

This embodies the ACP invariant: *ACP reports divergence. ACP does not resolve divergence.* Org-B cannot extend Org-A's `valid_until`, override Org-A's `delta_max`, or substitute its own snapshot for Org-A's signed record. Each institution's attestations are cryptographically bound to its own key and immutable after issuance.

7.3 Deployment

The demo ships with a single `Dockerfile` (parameterized by `ARG ORG=org-a|org-b`), a `docker-compose.yml` with health check, and a dedicated Go module with a `replace` directive pointing to `impl/go/`:

```
docker compose up --build    # from examples/multi-org-demo/
curl http://localhost:8081/request | jq .
```

8 Security Model

ACP explicitly defines which threats it mitigates, which properties it guarantees, and which risks fall outside its scope.

8.1 Threat Model (STRIDE)

Category	Threat	Mitigation in ACP
Spoofing	AgentID impersonation	AgentID = SHA-256(pk). Without valid signature → immediate DENIED.
Tampering	Token or event alteration	Ed25519 covers all fields. Chained ledger: altering one event invalidates all subsequent.
Repudiation	Agent denies executed action	ActionRequest digitally signed. Non-repudiation by design.
Info Disclosure	Capability exposure	Tokens reveal only the necessary subset. Channel confidentiality via TLS.
DoS	Request flooding	Rate limits per `agent_id`. Anomalous frequency control.
Elevation	Delegation that expands privileges	$C_{\text{delegated}} \subseteq C_{\text{original}}$. Cryptographically verified at each hop.

Table 4: STRIDE threat model and ACP mitigations.

8.2 Guaranteed Security Properties

ACP guarantees the following properties when the implementation is compliant:

- **Artifact integrity.** EUF-CMA security of Ed25519. Impossible to modify a token or event without invalidating the signature.
- **Identity authenticity.** Only whoever possesses `sk` can generate a valid signature under the corresponding `pk`.
- **No privilege escalation via delegation.** Demonstrable by induction over the delegation chain.
- **Anti-replay.** The single-use challenge makes reusing a proof of possession ineffective.
- **Effective revocation.** $Valid(t) = valid\_sig \land \neg expired \land \neg revoked \land valid\_delegation$. All four conditions must be true simultaneously.

8.3 Declared Residual Risks

ACP explicitly declares what it cannot resolve:

- **Total compromise of the RIK held in HSM.** ACP defines the emergency rotation process but cannot prevent a physical compromise of the custody infrastructure.
- **Coordinated institutional collusion.** If multiple institutions act maliciously in a coordinated manner, they can generate valid artifacts. ACP guarantees traceability, not prevention of malicious agreements between parties.
- **Implementation failures.** ACP is a specification. An implementation that violates prohibited behaviors can compromise all protocol guarantees. Conformance requires formal testing.
- **ITA bootstrap.** The only point that depends on an out-of-band channel. Once the ITA root key is resolved, everything subsequent is autonomous.

9 Conformance and Interoperability

9.1 Conformance Levels

Level	Required documents	Enabled capability
L1 — CORE	ACP-SIGN-1.0, ACP-CT-1.0, ACP-HP-1.0	Token issuance with cryptographic P
L2 — SECURITY	L1 + ACP-RISK-1.0, ACP-REV-1.0, ACP-ITA-1.0	Risk eval, transitive revocation, ITA
L3 — FULL	L2 + ACP-API-1.0, ACP-EXEC-1.0, ACP-LEDGER-1.3	Complete verifiable auditing
L4 — EXTENDED	L3 + ACP-PAY-1.0, ACP-NOTIFY-1.0, ACP-DISC-1.0, ...	Full governance suite
L5 — DECENTRAL.	L4 + ACP-D suite	Federation without central ITA

Table 5: ACP-CONF-1.2 conformance levels.

9.2 Conformance Declaration

Every compliant implementation MUST expose a public endpoint without authentication:

```
GET https://<endpoint>/acp/v1/conformance
```

This endpoint returns the achieved level, implemented documents, declared extensions, and declaration date. It allows any external partner to verify the conformance level of a counterparty before establishing an ACP relationship.

9.3 Prohibited Behaviors

ACP defines 12 behaviors that no compliant implementation can exhibit. If any is exhibited, the implementation cannot declare conformance at any level:

Code	Prohibited behavior
PROHIB-001	Approving a request when any evaluation component fails
PROHIB-002	Reusing an already-consumed Execution Token
PROHIB-003	Omitting signature verification on any incoming artifact
PROHIB-004	Treating a not-found `token_id` as active in a revocation context
PROHIB-005	Allowing state transition from `revoked`
PROHIB-006	Issuing an ET without a prior APPROVED AuthorizationDecision
PROHIB-007	Modifying or deleting Audit Ledger events
PROHIB-008	Silencing ledger corruption detection
PROHIB-009	Ignoring `max_depth` in delegation chains
PROHIB-010	Implementing an offline policy more permissive than ACP-REV-1.0
PROHIB-011	Approving requests from agents with `autonomy_level` 0
PROHIB-012	Continuing to process an artifact with an invalid signature

Table 6: ACP prohibited behaviors. Exhibiting any disqualifies conformance at all levels.

9.4 B2B Interoperability Conditions

- **L1 Interoperability:** Institution A can verify tokens from B if both implement ACP-CONF-L1, A has access to B's public key (via ITA or out-of-band), and B's tokens use ACP-SIGN-1.0 algorithms.
- **L2 Interoperability:** A can delegate to B's agents if both implement ACP-CONF-L2, are registered in a common ITA or with mutual recognition, and B's revocation endpoint is accessible to A.
- **L3 Interoperability:** A can audit B's ledger if B implements ACP-CONF-L3, A can resolve B's public key via ITA, and B exposes `GET /acp/v1/audit/query`.

10 Use Cases

ACP is sector-agnostic. The mechanisms are identical regardless of industry. What varies is the configuration of capabilities, resources, and autonomy levels.

10.1 Financial Sector — Inter-Institutional Payment Agents

ACP-PAY-1.0 extends the capability registry with formal specifications for `acp:cap:financial.payment` and `acp:cap:financial.transfer`. Mandatory constraints in the token include `max_amount` and

`currency`. 12 specific validation steps cover limit verification, beneficiary validation, and time window control. In a financial B2B scenario, Bank A can authorize an agent to execute payments up to a defined amount to pre-approved beneficiaries, with a complete record verifiable by Bank B without needing shared proprietary systems.

10.2 Digital Government — Document Processing

Government agents that process documents can operate under ACP with `autonomy_level` 1 or 2, requiring human review for any action with a Risk Score above the configured threshold. Ledger traceability provides forensic evidence for regulatory audits and transparency processes.

10.3 Enterprise AI — Multi-Company Orchestration

In agent pipelines that cross organizational boundaries, ACP allows each organization to maintain formal control over what other organizations' agents can do in their systems. Chained delegation allows an agent in company A to operate in company B's systems with explicitly delegated capabilities, without B needing to trust A's internal controls—only the chain of signed tokens.

10.4 Critical Infrastructure — Monitoring and Actuation Agents

For systems where an incorrect action has irreversible consequences, `autonomy_level` 0 ensures any actuation request is DENIED without evaluating the Risk Score. The ledger provides the forensic record needed for post-incident analysis.

11 Specification and Implementation Status

ACP v1.15 is a complete Draft Standard specification with a full Go reference implementation.

11.1 Active Specifications — v1.15 (36 documents)

11.2 Reference Implementation — 22 Go Packages + Multi-Org Demo

The Go reference implementation in `impl/go/` covers all L1–L4 conformance levels. All 22 packages pass `go test ./...`. A Python SDK (`impl/python/`) covers the ACP-HP-1.0 handshake and all ACP-API-1.0 endpoints. The `examples/multi-org-demo/` directory (GAP-14) provides a runnable two-organization scenario using the real `pkg/policyctx` and `pkg/reputation` packages; see Section 7.

11.3 Conformance Test Vectors — 73 Signed Vectors

The `compliance/test-vectors/` directory contains 73 signed test vectors per ACP-TS-1.1:

Level	Document	Title
L1	ACP-SIGN-1.0	Serialization and Signature
L1	ACP-AGENT-1.0	Agent Identity
L1	ACP-CT-1.0	Capability Tokens
L1	ACP-CAP-REG-1.0	Capability Registry
L1	ACP-HP-1.0	Handshake / Proof-of-Possession
L1	ACP-DCMA-1.1	Delegated Chain Multi-Agent
L1	ACP-MESSAGES-1.0	Wire Message Format
L1	ACP-PROVENANCE-1.0	Authority Provenance
L2	ACP-RISK-1.0	Deterministic Risk Engine
L2	ACP-REV-1.0	Revocation Protocol
L2	ACP-ITA-1.0	Institutional Trust Anchor
L2	ACP-ITA-1.1	ITA Mutual Recognition
L2	ACP-REP-1.2	Reputation Module
L3	ACP-API-1.0	HTTP API
L3	ACP-EXEC-1.0	Execution Tokens
L3	ACP-LEDGER-1.3	Audit Ledger (mandatory institutional sig)
L3	ACP-PSN-1.0	Policy Snapshot
L3	ACP-POLICY-CTX-1.1	Policy Context Snapshot
L3	ACP-LIA-1.0	Liability Attribution
L4	ACP-HIST-1.0	History Query API
L4	ACP-PAY-1.0	Financial Capability
L4	ACP-NOTIFY-1.0	Event Notifications
L4	ACP-DISC-1.0	Service Discovery
L4	ACP-BULK-1.0	Batch Operations
L4	ACP-CROSS-ORG-1.1	Cross-Organization Bundles
L4	ACP-GOV-EVENTS-1.0	Governance Event Stream
L4	ACP-REP-PORTABILITY-1.1	Reputation Snapshot Portability
Gov.	ACP-CONF-1.2	Conformance — sole normative source
Gov.	ACP-TS-1.1	Test Vector Format
Gov.	RFC-PROCESS	Specification Process
Gov.	RFC-REGISTRY	Specification Registry
Gov.	ACR-1.0	Change Request Process

Table 7: ACP v1.15 active specification documents by conformance level (33 primary documents shown). The remaining 4 documents are the ACP-D decentralized suite (L5, design phase) not yet published. Superseded versions (CONF-1.0, CONF-1.1, LEDGER-1.2, REP-1.1, AGENT-SPEC-0.3, DCMA-1.0, CROSS-ORG-1.0, REP-PORTABILITY-1.0) archived in `archive/specs/`.

11.4 Roadmap

12 How to Implement ACP

ACP is a specification—it does not require adopting any specific platform. It can be implemented on top of existing infrastructure.

Package	Spec	Level
`pkg/crypto`	ACP-SIGN-1.0 + ACP-AGENT-1.0	L1
`pkg/tokens`	ACP-CT-1.0	L1
`pkg/registry`	ACP-CAP-REG-1.0	L1
`pkg/handshake`	ACP-HP-1.0	L1
`pkg/delegation`	ACP-DCMA-1.1	L1
`pkg/provenance`	ACP-PROVENANCE-1.0	L1
`pkg/risk`	ACP-RISK-1.0	L2
`pkg/revocation`	ACP-REV-1.0	L2
`pkg/reputation`	ACP-REP-1.2 + ACP-REP-PORTABILITY-1.1	L2/L4
`pkg/execution`	ACP-EXEC-1.0	L3
`pkg/ledger`	ACP-LEDGER-1.3	L3
`pkg/psn`	ACP-PSN-1.0	L3
`pkg/policyctx`	ACP-POLICY-CTX-1.1	L3
`pkg/lia`	ACP-LIA-1.0	L3
`pkg/hist`	ACP-HIST-1.0	L4
`pkg/govevents`	ACP-GOV-EVENTS-1.0	L4
`pkg/notify`	ACP-NOTIFY-1.0	L4
`pkg/disc`	ACP-DISC-1.0	L4
`pkg/bulk`	ACP-BULK-1.0	L4
`pkg/crossorg`	ACP-CROSS-ORG-1.1	L4
`pkg/pay`	ACP-PAY-1.0	L4
`cmd/acp-server`	ACP-API-1.0	L3

Table 8: Go reference implementation packages. All pass `go test ./...`.

Suite	Positive	Negative	Spec
CORE (SIGN, CT, HP)	4	4	L1
DCMA	2	2	L1
HP	2	8	L1
LEDGER	3	8	L3
EXEC	2	7	L3
PROV	2	7	L1
PCTX	4	9	L3
REP	3	6	L4
Total	**22**	**51**	

Table 9: Conformance test vector suites. All positive vectors carry real Ed25519 signatures (RFC 8037 Test Key A) and real SHA-256 hash chains.

12.1 Minimum Requirements for L1 Conformance

- **Ed25519 public key infrastructure.** Key pair for each agent.
- **JCS implementation (RFC 8785).** Deterministic canonicalization for all signed artifacts.
- **Capability Token issuance and verification** with all mandatory fields per ACP-CT-1.0 §5.
- **Handshake endpoint** to issue and verify challenges (ACP-HP-1.0 §6).
- **Capability registry** with the core domains of ACP-CAP-REG-1.0.

Item	Status
Core specs (L1–L4), 36 documents	Complete
Go reference implementation (22 packages)	Complete
Conformance test vectors (73 signed)	Complete
OpenAPI 3.1.0 (`openapi/acp-api-1.0.yaml`, 17 endpoints)	Complete
Python SDK (`impl/python/`)	Complete (L1 + full API client)
Docker image (`ghcr.io/chelof100/acp-server:latest`)	Complete
Multi-org interoperability demo (`examples/multi-org-demo/`)	Complete (GAP-14)
L5 Decentralized (ACP-D)	Specification in design (v2.0)
Post-quantum algorithm migration	Research phase
IETF RFC submission	After L5 stabilization

Table 10: ACP v1.15 development roadmap.

12.2 Additional Requirements for L3 Conformance

- Root Institutional Key held in HSM with documented rotation process.
- Registration with an ITA authority (centralized or federated model).
- Deterministic risk engine with the four factors of ACP-RISK-1.0.
- Revocation endpoint (Mechanism A: online endpoint, or Mechanism B: CRL).
- Append-only storage for the Audit Ledger with per-event signing.
- Complete HTTP API per ACP-API-1.0, including health and conformance endpoints.
- Public conformance declaration at `GET /acp/v1/conformance`.

12.3 What ACP Does Not Prescribe

ACP defines the *what*—mechanisms, flows, data structures, requirements. It does not prescribe the *how* of internal implementation: programming language, database, HSM provider, ITA provider, or specific integration with existing RBAC or Zero Trust systems.

13 Conclusion

Autonomous agents are already operating in institutional environments. The question is not whether they will operate—it is whether they will do so with or without formal governance. ACP proposes that they operate with formal governance, with verifiable mechanisms, and with traceability that can withstand an external audit.

ACP is not the first attempt to control autonomous agents. It is the first attempt to do so through a formal technical specification with precise state models, demonstrable security properties, and verifiable conformance requirements. The difference between a best-practices policy and a formal protocol is exactly that: behaviors are defined, failures have specific error codes, and conformance can be verified.

The goal of ACP is not to make agents more capable. It is to make them governable. That is a necessary condition for their institutional deployment to be sustainable at scale.

ACP does not eliminate all sources of failure in distributed agent systems. It does not prescribe network transport, consensus mechanisms, or business-level dispute resolution. What it provides is

a structured, auditable, and formally specified model for admission control, delegation, revocation, and cross-organization interaction— the layer that existing identity and policy frameworks leave unaddressed.

The v1.15 specification is complete. A full Go reference implementation (22 packages, L1–L4), 73 signed conformance test vectors, an executable multi-organization interoperability demo (`examples/multi-org-demo/`), and an extended OpenAPI 3.1.0 specification (17 endpoints) are publicly available at https://github.com/chelof100/acp-framework-en.
Preprint: https://arxiv.org/abs/2603.18829 **Zenodo:** 10.5281/zenodo.19150239
The specification and implementation are open for technical review, pilot implementation, and formal standardization.

TraslaIA invites organizations interested in adopting ACP, contributing to its evolution, or participating in the standardization process to reach out directly.

Marcelo Fernandez | TraslaIA
info@traslaia.com | https://agentcontrolprotocol.xyz

A Glossary

Term	Definition
AgentID	Cryptographic identifier: `base58(SHA-256(Ed25519_public_key))`. Immutable and unforgeable.
Capability Token (CT)	Signed JSON artifact authorizing an agent to perform specific actions on a defined resource during a limited period.
Execution Token (ET)	Single-use artifact issued after an APPROVED decision. Authorizes exactly that action at that moment.
ITA	Institutional Trust Anchor. Authoritative registry linking `institution_id` to institutional Ed25519 public key.
RIK	Root Institutional Key. Institution's Ed25519 key pair held in HSM.
Risk Score (RS)	Integer in $[0, 100]$ produced by the deterministic risk function. Determines the authorization decision.
Autonomy Level	Integer 0–4 assigned to an agent determining applicable risk evaluation thresholds.
PoP	Proof-of-Possession. Cryptographic proof that the bearer of a CT possesses the corresponding private key.
Audit Ledger	Chain of signed events where $h_n = \text{SHA-256}(e_n \| h_{n-1})$. Append-only and immutable.
MRA	Mutual Recognition Agreement. Bilateral document signed by two ITA authorities for cross-authority interoperability.
ESCALATED	ACP decision when RS is in the intermediate range. Action not executed until explicit resolution.
Fail Closed	Design principle: on any internal failure, the action is denied. Never approved by default.

Table 11: ACP Glossary.

Acknowledgements

The ACP specification emerged from analysis of operational challenges in deploying autonomous agents in regulated B2B environments. The protocol design was informed by the IETF RFC process, the Kubernetes admission control architecture, and the SPIFFE/SPIRE workload identity model.

References

[1] Dick Hardt. The OAuth 2.0 authorization framework. RFC 6749, Internet Engineering Task Force, October 2012. IETF RFC 6749.

[2] Simon Josefsson and Ilari Liusvaara. Edwards-curve digital signature algorithm (EdDSA). RFC 8032, Internet Engineering Task Force, January 2017. IETF RFC 8032.

[3] Open Policy Agent Contributors. Open policy agent, 2024. Policy evaluation engine. ACP-RISK-1.0 Step 3 is compatible with OPA as backend.

[4] Anders Rundgren, Bret Jordan, and Samuel Erdtman. JSON canonicalization scheme (JCS). RFC 8785, Internet Engineering Task Force, June 2020. IETF RFC 8785.

[5] SPIFFE Project. SPIFFE / SPIRE: Secure production identity framework for everyone, 2024. Cryptographic workload identity. ACP builds on SPIFFE identity to add capability scoping.