Main Paper

Agentic Authorization Delegation Spec Overview

Introduction

This report evaluates authorization and access control specifications specifically through the lens of agentic permissions and delegation.

Why agents specifically? Conventional access control was already strained by ordinary human sharing, but LLM-based agents completely break its assumptions. Agents are often short-lived, sometimes existing for only seconds, therefore provisioning each agent with accounts or ACL entries doesn't scale, and an agent's own identifier can never be the responsible party. Responsibility has to trace back to the principal operator who gave it the task. Agents can't be trusted to keep secrets (a clever prompt may extract a credential), and, unlike humans, they can't be deterred by post-facto punishments or personal consequences. In cases where conventional software has a defined API, an agent's behavior is effectively unbounded: the permissions it will need can't be fully known in advance, making "least privilege" a dynamic problem rather than a provisioning-time one. These properties push the authors toward forms of authorization that are delegated, attenuated, and verifiable at request time, rather than derived from an identity. Agentic AI's fundamental risks suggest that any proof-of-possession keys must be held outside the agent itself (by a more deterministic component trusted to use them on the agent's behalf).

To address the problem of Agentic authorization and delegation holistically, we assessed specifications addressing various authorization layers. Their strengths and weaknesses are addressed primarily against their scope and goals, with an eye to their various possible recombinations and configurations. A given spec may address one or several of the following:

- Data model: how the various mandatory and optional data elements of an authorization get expressed.
- Protocol: how tokens are requested, presented, and (optionally) revoked.
- Possession mechanism: how tokens are secured, e.g. through some proof of possession (and specifically protocol for agentic management of keys and token, which are sometimes profiled externally to the specification itself).
- Chainable proof methods: how a multi-hop chain of delegations is expressed and verified.

Delegation is central to the problem of agentic authorization because the only alternative to delegation is impersonation. Delegation can be done without capabilities, using identity-based (authentication-centric) systems, but that entails some degree of ambient authority, which opens the door to “confused deputy” attacks, whereby too much faith is placed in fallible (or second-hand) authentication. Cedar and similar systems illustrate the point: they support more flexible delegation while remaining identity-based, so they remain subject to the confused deputy problem, and they also place policy evaluation on the critical path of every request, incentivizing attacks on the identity layer rather than the policy layer. The same applies one layer down, at the decision protocol. AuthZEN standardizes how an enforcement point asks a policy decision point for a verdict, but the authority consulted still lives at that server and its trust model is still based on identity.

A quick note on the confused deputy problem, which was central to our motivation and inquiry here. Typically, people refer to the slippage between logged identities and actual-executing, actually-liable parties as a confused-deputy situation, but we took a more existential approach: we think of impersonation and credential-sharing and exceptions to rigid logging or access policies as invitations, which make authentication and credential data the weakest links. Agents are nothing if not expert weakest-link finders! To foreclose the possibility of confused deputies, we maintain, designation and authorization must be combined, explicitly stating which actions are allowed to be performed on which resource, verb, or API endpoint, rather than leaving that to be solved indirect via authentication and policy. This combination is definitive of a family of approaches generally grouped under the category of "capability systems", explicitly formalizing capabilities, as opposed to systems relying on ambient authority (e.g. assigning authority by identity and/or role).

Capability systems can be categorized by which parts of the system do the enforcement. The main types are:

- hardware capabilities (the oldest form)
- object references enforced by the platform (which require a memory-safe language to be securable at scale)
- opaque tokens
- certificates
- cryptographic-only (encryption-based) systems.

In certificate systems, metadata is bundled *with* the permission and presented together with it; in opaque token, platform, and hardware systems, the metadata is stored somewhere else and

merely referenced. Certificate systems themselves come in two types: bearer tokens and proof of key possession.

Multi-hop (chainable) delegation is important. Without it, after a "single hop" delegation (traditional OAuth2), the system lends itself to impersonation (for example, copy-and-pasting an OAuth2 token into an agent's prompt or context window for it to act as you in the system). In this increasingly common scenario, the audit trail is degraded, and it's not possible to revoke the agent's privilege without revoking the human's privilege as well. At scale, this is an untenable workaround.

Attenuation, or the ability to delegate a smaller scope of capabilities, is an essential part of the picture. Delegation should enforce the **principle of least privilege,** giving an actor working on your behalf as little privilege (and incurring as little risk) as necessary. When enlisting help from an agent, the user should give it only the minimum permissions necessary to perform the task. Attenuation enforced by the capability system itself prevents escalation attacks, which were less frequent and existential a risk when enterprise-internal systems were only being used by human employees.

Traditional capability systems are intentionally not identity-based, and in many cases can be configured to be almost identity-blind. For agentic use cases, however, it's important to constrain identifier choices (and maintain any maps or indirections necessary) so as to maximize accountability and logging, *particularly* because agentic use-cases involve so many liable parties, with non-determinism and risk distributed across so many components. Complicating identifier choices and classical best practices are the inherent challenges of cross-domain, cross-organizational identifiers, since agents often perform tasks across different web domains, trust zones, and execution contexts, which may in turn require partial confidentiality or bring their own logging and authentication requirements. Given all of the above, we take the strong stance that bearer tokens are dangerous to use at any layer of such a system, due to the ease of replay attacks and their tendency to appear in logs; luckily, this position is becoming less controversial over time across the security and identity communities, as more and more research and security incidents indicate the near- or total impossibility of trusting frontier models with such systems.

Specification Evaluation Checklist

In this report we evaluate major authorization and/or access-control specifications that address any of the design questions listed in the Introduction: primarily, we study how delegation is supported by the data models, invocation protocols, and chainable proof methods, but also

other aspects of protocol and logging are addressed as well. Validity checks must be done at the time of request (by a component trusted by the resource provider), and there is much room for configuration and profiling across all these systems, but we have focused on the core affordances of these specifications at a high-level.

The assessment criteria largely track Stiegler's seven aspects of sharing (see the Delegated Authority Problem-space report), with a few adaptations for spec evaluation. The adaptations are:

- Composability is not listed as a separate criterion, since any capability system satisfies it trivially, so it does not discriminate among the candidates under evaluation;
- Functional resistance to confused deputy attacks (item 2) replaces the more formal composability, as it was proved the more useful rubric,]
- An additional requirement that authorization *policies* be representable (at least by reference) in the specification (item 3), which is a practical necessity in enterprise and large-scale deployments, and
- Stiegler's "dynamic" aspect (delegation without undue delay) is relaxed from a requirement to the offline-capable nice-to-have below.

For this report, the final list of assessment criteria is:

1. **Accountability:** Specification must allow stable, in-band differentiation between the agent and its principal operator (e.g., natural or legal person)
2. **Confused-deputy Resistance:** relative infeasibility of impersonation and authentication attacks, mostly from reducing “ambient authority.”
3. R**epresentability of authorization policies** (either by embedding or linking to them)
    a. This includes **functional policies** like "do not delegate further" or “this permission is non-delegatable”
    b. This includes **re-delegation policies** like “only delegate within X boundary or role”
    c. "**Please**": A request containing desired rules for redelegating but not refusing to honor violations.
    d. "**Disclosure governance**": Confidentiality of tokens and logs is not just configurable but governable.
4. **Chainability**: Authorization must be able to be delegated: At root, authorization is delegated from principal/operator to agent. But also, agents must be able to delegate to further subsystems or other agents.
5. **Cross-organizational execution** (validation checks are independently verifiable, locally verifiable). At minimum, different organizations/groups should be able to express authorization policies without having accounts on each other's IAM systems.
6. **Attenuated delegation**: delegation of a subset of permissions as a first-class data element.

7. **Self-revocability:** The capability holder, somebody in the delegation chain, or a holder of an explicit revocation capability, must be able to revoke a delegated permission, modulo network partitions.

Not required but nice to have:

8. **Authentication grounded in first-class, in-band Proof of Possession**
9. **Privacy of Delegation chain:** Can the system hide some or all the delegation data and metadata in the proof chain from agent / token carrier, and/or from intermediate counterparties?
10. **Offline-capable:** Create tokens offline, delegate offline, use/present offline. Agentic Use cases: most of them online, but some are offline-capable to (local machine, or local LAN only agentic use cases). Also, offline-capable parts of the spec can offer efficiency advantages. For example, if it's possible to delegate without having to contact the resource server, there's a significant speed advantage.

General Data Model (In Common)

Despite differences in encoding and terminology, the data models under evaluation share a common shape. A delegated permission tends to carry the following fields:

- "agent"—how the agent is identified (most often a DID or a key descriptor).
- "principal"—who this permission is on behalf of.
- "resource"—the resource the permission is given for.
- "actions" —actions/operations allowed on that resource.
- "caveats" / "conditions"—additional restrictions and conditions on use.
- "delegation chain"—the cumulative record of the upstream delegations made in a multi-hop delegation system.

Alongside these fields, each spec also addresses several mechanisms that sit partly in the data model and partly in the protocol:

- "protocol"—how to request this permission.
- "revocation"—how a permission is revoked (mostly protocol, though revocation has a data model aspect as well, such as pointing to where revocation status is checked).
- "invocation" (separate protocol and data model)—how to invoke a permission for a given resource.
- cryptographic suites (if applicable)—how to prove cryptographic key possession for delegation and invocation.

A table mapping each spec's concrete field names onto this common shape appears in the Synthesis chapter, under Data Model Correspondence.

A Note on Terminology

To discuss delegation, we need to at least have names for the parties involved. We use the following terms:

- Delegat**or**:The party who is *doing* the delegating. The source of the authorization. This one is easy, most texts agree on this one.
- Delegat**ee**:The party who *is being delegated to*. The target of the delegation. This term is harder. Some texts also refer to this party as a “delegate” (noun). This makes sense, but in written form, it's easy to confuse with 'delegate' (verb). Some texts even use “delegee” for this party. For ease of use, this report standardizes on 'delegat**ee**'.

Of course, these roles are transitive, and a delegatee becomes in turn the delegator the moment they sub-delegate whatever permission or authorization they have to yet another party in the chain.

OAuth: Hardening, Consolidation, and the Push Toward Autonomous Authorization

This chapter covers what is actually deployed today—OAuth 2.0 (RFC 6749/6750), as hardened by RFC 9700 (2025 Security Best Current Practice)—then traces the evolution toward OAuth 2.1 and, separately, the independent AAuth draft.

Each section below opens with an italicized tag, one of:

- Deployed standard
- Adjacent RFC/profile
- OAuth 2.1—consolidation draft
- AAuth—independent draft

These standards clarify the most common solution in use for Google, Okta, Auth0, GitHub, or a typical enterprise IdP today, versus what's still in development and limited/niche deployments.

Overview

OAuth (Open Authorization) is an open standard for **access delegation** anchored in HTTP calls, session state, and redirects. It allows users to grant third-party applications secure, scoped access to their data hosted on another service without ever handing over their passwords. Much like a "valet key" for a car, which lets a valet drive and park the vehicle but prevents them from unlocking the glovebox or trunk, OAuth uses limited-scope, session-bound **Access Tokens** rather than shared passwords or other static, long-lived credentials to protect user data.

The standard running in production today—what every major identity provider implements for most clients at scale— is OAuth 2.0: RFC 6749 (the core authorization framework) and RFC 6750 (bearer token usage), both from 2012, with additional security on higher layers. OAuth 2.0 is very much a framework, not a fixed protocol: it deliberately leaves many security-relevant decisions to implementers, architects, and configuration. This flexibility is precisely why a decade of real-world attacks produced a long list of follow-on RFCs tightening specific gaps (PKCE, JWT access tokens, token exchange, sender-constraining, DPoP, and so on).

In January 2025, the IETF published RFC 9700, "Best Current Practice for OAuth 2.0 Security," which consolidates that decade of lessons into concrete guidance: don't use the Implicit grant, don't use Resource Owner Password Credentials, always use PKCE where possible, match redirect URIs exactly, and prefer sender-constrained tokens. RFC 9700 is guidance on top of RFC 6749/6750. It doesn't supersede or deprecate them.

OAuth 2.1 (draft-ietf-oauth-v2-1) is the next step, versioning the protocol rather than just profiling it. It is an attempt to fold RFC 9700's hardening directly into a single normative document that deprecates RFC 6749/6750. As of this writing, it is still an Internet-Draft in its seventh year of iteration. "OAuth 2.1" is best understood as the direction the ecosystem is consolidating toward, with most major 2.0 deployments largely compliant with minor changes, not a separately deployed protocol version.

Separately, AAuth (draft-hardt-oauth-aauth-protocol) is an independent, early-stage draft built specifically for agent-to-resource access, designed to coexist with OAuth and OIDC, not to extend either.

Part 1: What's Actually Deployed: OAuth 2.0 plus RFC 9700

The Core Framework

Deployed standard. RFC 6749 defines the four classic grant types (Authorization Code, Implicit, Resource Owner Password Credentials, Client Credentials) plus Refresh Tokens. RFC 6750 defines how bearer tokens are presented to a resource server. Some form of Client Registration is assumed but out of scope of RFC 6749; many later RFCs extend alternate mechanism. This is the substrate nearly every production OAuth deployment is built on.

RFC 9700: The Hardening Layer in Force Today

Deployed standard / BCP. RFC 9700 is the document that defines a secure OAuth 2.0 deployment.

RFC 9700 mandates the following:

- Don't use the Implicit grant (response_type=token). Tokens returned in a URL fragment are exposed to browser history, referrer leakage, and any script on the page.
- Don't use Resource Owner Password Credentials. Password-based delegation trains users to type credentials into third-party UIs and gives the client raw password access.
- Use PKCE (RFC 7636) on the Authorization Code flow for all clients, not just public ones. PKCE it closes authorization-code-interception attacks that affect confidential clients too.
- Match redirect URIs exactly; no substring or wildcard matching.
- Prefer sender-constrained tokens (DPoP or mutual TLS) over plain bearer tokens where feasible.
- Use state for CSRF protection and PKCE's code_verifier together. Each protect against different attacks.

None of this requires specific use of "OAuth 2.1" as a separate thing to implement. OAuth 2.1's contribution is mostly editorial consolidation: combining these BCP recommendations and making the safe behaviors the only correct specified behaviors.

Client Authentication: Required vs. Common Practice

Deployed standard / Adjacent RFC. RFC 6749 only requires that confidential clients authenticate to the token endpoint somehow. client_secret_basic and client_secret_post are the baseline methods, widely used today.

Asymmetric client authentication has become common best practice for higher-assurance deployments:

- private_key_jwt (RFC 7523). The client signs an assertion with its own private key instead of sending a shared secret.
- Mutual TLS client authentication (RFC 8705). The client authenticates via its TLS certificate.

These are additions layered onto OAuth 2.0, not outright mandates of RFC 9700 or OAuth 2.1.

Proof of Possession: DPoP and mTLS

Adjacent RFC. By default, OAuth 2.0 bearer tokens (RFC 6750) are bearer instruments. Whoever holds the token can use it.

Two specs address this:

- DPoP (RFC 9449). The client generates a key pair and signs a proof bound to each request's HTTP method and URI. A stolen token is useless without the matching key.
- Mutual TLS (RFC 8705). Binds the token to the client's TLS certificate.

RFC 9700 recommends sender-constraining. It doesn't require it, and most OAuth 2.0 deployments still issue plain bearer access tokens.

Token Format: Opaque vs. Self-Contained

Deployed standard / Adjacent RFC. OAuth access tokens can be opaque (validated via token introspection, RFC 7662) or self-contained JWTs (verifiable offline per the JWT Access Token profile, RFC 9068). The choice is left to implementers by RFC 6749.

Revocation

Deployed standard / Adjacent RFC. RFC 7009 allows a client or resource owner request explicit revocation. Short-lived access tokens provide a passive backstop. OAuth 2.0 has no native notion of delegation-chain revocation, because it has no native delegation chain in the first

place. Delegation semantics are difficult to expose to the user, but delegations can be tracked across the logs of the Authorization Server.

Part 2: OAuth 2.1 : Where the Consolidation Is Heading

OAuth 2.1 consolidation draft. OAuth 2.1 (draft-ietf-oauth-v2-1) takes everything in Part 1's hardening layer and includes it in the core spec.

Key changes in OAuth 2.1 vs. today's hardened OAuth 2.0:

- Implicit and ROPC grants are removed from the spec text, not just discouraged.
- PKCE is required, not recommended, for every client on the Authorization Code flow.
- Redirect URI exact-matching is specified, not left as guidance.
- Sender-constrained tokens (DPoP/mTLS) remain a SHOULD, not a MUST. OAuth 2.1 does not mandate proof-of-possession tokens.
- Client secrets are still supported. The draft explicitly requires AS support for clients sending a plain client_secret. Asymmetric client authentication is not the OAuth 2.1 baseline.

A well-hardened, RFC 9700-compliant OAuth 2.0 deployment today is very close to "OAuth 2.1 compliant". OAuth 2.1's main value is editorial: a single document new implementers can build against without needing to piece together a decade’s worth of separate BCP RFCs.

Part 3: AAuth: A Separate, Independent Draft for Agent-to-Resource Access

AAuth—independent draft. Caveat: this is a single-author Internet-Draft, not an IETF working-group product. It has already been renamed and restructured more than once. Treat the below as a snapshot of a moving target.

Why It Exists

AAuth (draft-hardt-oauth-aauth-protocol) was created by an OAuth 2.0/2.1 co-author who concluded that OAuth, in any version, is a poor fit for agent-to-resource access because:

- Agents need authorization decisions made mid-task against resources they've never interacted with before.

- Agents have no durable client_id that travels across organizations the way OAuth assumes.

- Copying long-lived API keys into autonomous workloads recreates the shared-secret leak surface OAuth 2.0 and 2.1 are trying to move away from.

AAuth is explicitly designed to coexist with OAuth 2.0 and OpenID Connect, not extend either one.

Architecture

Roles (See Terminology, in newest draft):

- Person/Principal
- Agent Provider (AP): The Agent Provider is a centralized entity which controls which manages agent identity and issues agent tokens, providing centralized governance over a distributed agent fleet.
- Agent
- Person Server (PS): server that represents the principal to rest of the protocol. The person chooses their PS; it is not imposed by any other party. The PS manages missions, handles consent, asserts user identity, and brokers authorization on behalf of agents.
- Access Server (AS): A policy engine that evaluates token requests, applies resource policy, and issues auth tokens on behalf of a resource.
- Resource

Token types:

- `agent_token` establishes the agent's identity

- `resource_token` describes the access a resource needs

- `auth_token` grants access to a specific resource, where applicable

Core mechanism: Signed HTTP requests using HTTP Message Signatures, not bearer tokens trusted on presentation. The simplest mode requires no token issuance flow at all.

Four Resource Access Modes

AAuth defines four modes, increasing in complexity:

- Identity-based access: The agent signs requests with its agent_token. The resource verifies the signature and applies its own access policy. No authorization flow, no additional tokens. This replaces API keys with cryptographic identity.

- Resource-managed access (two-party): The resource handles authorization itself, optionally returning an opaque token for subsequent calls.

- PS-asserted access (three-party): A Person Server vouches for claims about the person on whose behalf the agent acts, with a consent step and an auth_token issued on approval.

- Federated access (four-party): Extends the three-party model across an additional trust boundary.

Key discovery: Agents and resources discover signing keys via well-known metadata and JWKS-style endpoints, similar in spirit to OAuth's JWKS discovery, but used for HTTP-signature key lookup, not bearer-token verification.

Part 4: Comparative Scorecard

Note: A comparative scorecard placing OAuth 2.0 and AAuth alongside the capability systems previously appeared here. It has been superseded by the merged scorecard in the Synthesis chapter, which derives its verdicts from each specification's own chapter.

Closing Note

In production today, almost all systems are using OAuth 2.0, hardened to RFC 9700's recommendations, with client secrets still alive and well at most token endpoints and bearer tokens still the default token type. OAuth 2.1 is where that hardening is headed as a single normative document. However, the changes are fairly minor. It doesn’t mandate PoP nor does it eliminate client secrets.

AAuth, meanwhile, isn't a future version of OAuth at all. It's a parallel, signature-based protocol built because its author judged some core assumptions of the OAuth model itself a poor match for autonomous agents. AAuth, however, pushes complexity and risk into a fiduciary agent. This super-agent manages delegations, token-swaps, and other intermediary security and trust decisions. While AAuth ceremonies are different than those in OAuth, most of the token formats are lifted directly from OAuth, making it more of a logical variant than a complete overhaul of OAuth. UCAN/zCAP-LD remain the only one of the three with native, offline, multi-hop delegation -- the clearest architectural line separating capability systems from the many HTTP Servers defining both the OAuth lineage and its AAuth variant (which adds a third Server).

AAuth and Delegation - In-depth analysis and Scorecard

Overview3

AAuth is an emerging authorization architecture designed for authenticated interactions among people, AI agents, resources, Person Servers (PS), and Agent Providers (AP). Rather than treating software agents as extensions of end users, AAuth introduces authenticated agents as first-class participants capable of invoking protected resources, receiving asynchronous events, and operating across organizational boundaries. Crucially, the “container” for these cross-organizational distributed transactions is called a “mission” (dovetailing with other work by OAuth Working Group insiders). While the previous chapter touched on AAuth in terms of its common assumptions and mechanisms, assessing AAuth as a bespoke protocol for managing and auditing agentic distributed transactions is easier treating it in isolation.

Unlike capability-based delegation systems such as UCAN and zCaps, AAuth does not represent delegated authority as a self-contained, cryptographically attenuated artifact carried entirely within a delegated credential. Instead, delegation is server-mediated. Authority is anchored within the Person Server, with delegation history represented through the act claim, Mission References, and the Mission Log maintained by the Person Server. AAuth therefore defines explicit delegation mechanics while intentionally separating delegation state from portable capability objects.

AAuth is evaluated here as a server-mediated delegation architecture. It authenticates participants, establishes trusted execution between them, records delegation history, and supports authenticated event-driven execution. Questions concerning delegation semantics, attenuation policy, authority evolution, and the policy logic behind execution-time decisions remain outside the protocol itself and are expected to be supplied by complementary architectural layers.

Three reference points are used:

- draft-hardt-oauth-aauth-protocol-10 (6 August 2026), together with draft-hardt-httpbis-signature-key-08, the companion specification to which AAuth defers key conveyance, discovery, and algorithm determination;

- The AAuth Events draft, which extends the architecture with authenticated asynchronous event delivery between resources and agents through Agent Providers; and

- public implementation discussions describing the intended architecture and deployment model.

Where protocol specification and implementation practices differ, this distinction is noted throughout the evaluation.

A note on method: This chapter was drafted with AI research collaborators under a gated review process, with final editorial authority held by the author. Load-bearing claims were verified directly against the AAuth specification repository rather than model recall — including the token-exchange question, the act claim's provenance, credential key-binding, and the Appendix B.3.7 attenuation rationale.

Scorecard

1. Accountable (agent vs. principal/operator)

Verdict: Yes

AAuth distinguishes authenticated people, agents, Person Servers, Agent Providers, and resources. Delegation history records acting parties through the act claim.

2. Resistant to confused deputy (no ambient authority)

Verdict: Partial

The resource-managed, PS-asserted, and federated modes bind an authorization to a designated resource and scope through the resource token, combining designation with authorization. The **optional** account parameter added in draft-10 narrows that binding further, naming which account at the resource the authorization covers and propagating through the resource token into the auth token. Identity-based access does not do this: the resource authorizes on the agent's identity alone and applies its own access control, which is ambient authority.

3. Represent authorization policies

Verdict: Partial

Authorization context and Mission References are represented, but expressive delegation policy—for example, undelegatable authority, advisory policy, or attenuation constraints—is intentionally out of scope of the protocol, or inside the logic and authority of the Person Server.

4. Chainable

Verdict: Yes

Delegation chains are represented through the act claim and the Mission Log. Chains are server-mediated rather than portable capability artifacts. Sub-agent nesting is single-level; deeper workflows use chained top-level agents, each an independent principal holding its own grant.

5. Cross-organizational / locally verifiable

Verdict: Yes

Designed for authenticated interactions across organizational boundaries while preserving local trust relationships.

6. Attenuated

Verdict: Partial — differs by mode

Sub-agent authorization supports attenuation: every request passes through the parent, which can refuse, attenuate, or rate-limit. Call chaining does not — downstream authorization is intentionally not required to be a subset of upstream scope. Cryptographically enforced attenuation is not intrinsic to the protocol in either case.

7. Self-revocable

Verdict: Partial

Token revocation, mission revocation, and propagation through a parent's grant are all specified. Revocation is exercised by the servers holding authority rather than by a holder acting on a credential in hand, and mission state administration beyond completion is deferred to a companion specification.

Authentication / Proof of Possession

Verdict: Yes

Core capability of the protocol. -10 requires a fully-specified alg identifier, recommends Ed25519, and prohibits none, symmetric algorithms, and the polymorphic EdDSA identifier that RFC 9864 deprecated.

Privacy of Delegation Chain

Verdict: Partial

Delegation history is maintained by the Person Server rather than being universally disclosed through portable credentials. The protocol records delegation continuity while allowing deployment architectures to determine how much of that history is exposed during execution.

Offline Capable

Verdict: Partial

Implementations MUST cache JWKS and SHOULD continue verifying against cached keys when a fetch fails, bounded by a cache lifetime of at most 24 hours, so token verification survives temporary loss of contact with an issuer. Obtaining authority remains online-only: AAuth's server-mediated model provides no offline delegation, and initial key discovery requires reachable metadata.

Detailed Evaluation

1. Accountable (agent vs. principal/operator)

AAuth explicitly distinguishes among authenticated people, AI agents, Person Servers, Agent Providers, and protected resources. Rather than treating software agents merely as extensions of user sessions, the protocol gives each participant its own authenticated identity and architectural role.

Delegated actions are represented through the act claim, allowing delegation history to be maintained across direct delegation, sub-agent delegation, and chained execution. Mission References further associate requests with the governing mission maintained by the Person Server.

Unlike certificate capability-based systems, accountability derives from authenticated protocol exchanges combined with server-maintained mission history rather than from cryptographically self-contained delegation credentials.

The specification also permits a resource to authorize an agent based solely on its identity, without interaction or mission context; working-group discussion has flagged that identity-only authorization reintroduces confused-deputy exposure, since the execution context rather than the requester's identity is what validates an action.

2. Resistant to confused deputy (no ambient authority)

The confused deputy problem arises where a party exercises its own permissions on a resource designated by someone else. The defence is to combine designation with authorization, so that what may be done is bound to what it may be done to.

In the resource-managed, PS-asserted, and federated modes AAuth does this. The agent obtains a resource token naming the resource and scope; the auth token issued against it carries that binding, and the resource enforces it. The OPTIONAL account parameter, added in -10, narrows the binding further: it names which account at the resource the authorization is for, drawn from the resource's own namespace, and is echoed through the resource token into the auth token, so an auth token for one account grants nothing at another.

Identity-based access is the exception, and a deliberate one. The agent signs requests with its agent token, and the resource applies its own access control based on who the agent is. There is no authorization flow and no designation carried with the request. The specification presents this as a replacement for API keys, which it is; but authority derived from identity alone is ambient, and an agent acting on a request supplied by another party has no way to signal that the request is not its own.

3. Ability to Represent Authorization Policies

AAuth represents authorization context through missions, authenticated requests, and protocol-defined execution relationships. Mission References bind requests to an existing mission while allowing authorization policy to remain under the control of the Person Server.

The protocol intentionally does not standardize a rich delegation policy language. Concepts such as undelegatable permissions, advisory delegation preferences, attenuation constraints, or policy composition remain responsibilities of external policy engines or governance frameworks.

This separation allows AAuth to remain focused on authenticated execution while supporting a wide variety of higher-level authorization models.

4. Chainable

AAuth provides explicit delegation mechanics through the act claim and the Mission Log. These mechanisms record delegation history across direct delegation, chained delegation, and sub-agent authorization.

Unlike certificate capability-based systems, delegation history is maintained by the Person Server rather than embedded entirely within a portable delegation credential. Verification

therefore depends upon authenticated interaction with the server maintaining mission state rather than validating an independently portable capability chain.

Accordingly, AAuth supports delegation chains, but those chains are server-mediated rather than self-contained cryptographic artifacts.

Sub-agent nesting is limited to a single level: a sub-agent must not have sub-agents of its own, and the specification enforces this from both directions. Deeper workflows are carried instead by call chaining, where each hop is an independent top-level agent holding its own grant rather than a recursive sub-agent — which is also why upstream-subset rules do not apply to it. What the specification defers to a companion document is mission state administration beyond completion: revocation state transitions, delegation-tree queries, and administrative interfaces. Delegation-chain lifecycle is not deferred; sub-agent revocation propagates through the parent's grant.

5. Cross-Organizational / Locally Verifiable

Supporting authenticated interactions across organizational boundaries is one of AAuth's principal architectural objectives.

Person Servers, Agent Providers, agents, and protected resources may participate across independent trust domains while preserving authenticated relationships between participants. Authentication evidence can therefore span organizations without requiring centralized identity management.

Trust remains anchored in authenticated protocol interactions and configured trust relationships rather than portable authorization credentials.

From a capability-based perspective, the need for Access Server federation may indicate that authority remains server-mediated rather than fully represented by the delegation credential itself. If the delegation credential carried sufficient authority for cross-domain verification, no ongoing relationship between authorization servers would be required.

6. Attenuated

Attenuation in AAuth differs by delegation mode. Sub-agent authorization supports it directly: every sub-agent request passes through the parent, which can refuse, attenuate, or rate-limit. Call chaining, by design, does not support delegation. Downstream authorization is intentionally not required to be a subset of upstream scope, and downstream scope is constrained by the

downstream resource's own policy together with Person Server evaluation against mission context.

Unlike UCAN or zCaps, attenuation is not represented as a cryptographically enforced property of a portable delegation artifact. Instead, attenuation depends upon server-managed mission state and policy evaluation.

The protocol therefore leaves attenuation semantics outside its definition, supplying contextual evaluation in place of algebraic constraint.

This is deliberate design rather than omission. AAuth does not adopt the RFC 8693 token-exchange model for attenuation (it borrows only that RFC's act claim structure for representing delegation chains), and Appendix B.3.7 states the rationale directly: downstream scope is intentionally not required to be a subset of upstream scope, because "the PS evaluates each hop against the mission context, providing governance-based constraints... more flexible than algebraic attenuation rules." The one place attenuation exists within an agent's own purview is sub-agent authorization, where the parent "can refuse, attenuate, or rate-limit" the sub-agent's authority.

7. Self-Revocable

AAuth includes mechanisms for token revocation and lifecycle management.

-10 specifies named revocation scenarios: a Person Server revoking an auth token it issued or provided, an Access Server revoking one it issued, an agent provider revoking an agent token, and a Person Server revoking a mission, after which subsequent token requests referencing that mission are denied. Revocation also propagates structurally. Revoking a parent's grant causes the next sub-agent authorization to fail, and tokens already issued expire within the hour.

The specification is candid about the limits. Verifying an auth token does not consult its issuer, so nothing in the verification path reports that a token has been revoked; a party that no revocation request reaches is bounded only by token lifetime. What remains unspecified is mission state administration beyond completion, deferred to a companion specification. Revocation is therefore exercised by the servers holding authority rather than by a holder acting on a credential in hand.

Summary

AAuth represents a distinct approach to delegated authorization for AI systems. Rather than expressing delegated authority as portable cryptographic capabilities, it anchors authority within the Person Server and maintains delegation continuity through authenticated protocol exchanges, mission state, and recorded delegation history. In the taxonomy of capability models, AAuth authorizations are neither bearer capabilities nor certificate capabilities: they are server-mediated proof-of-possession credentials, and the specification assigns multi-hop safety to mission-context evaluation rather than to algebraic attenuation, expecting complementary layers to supply the latter where required. Delegation in AAuth references stable agent identifiers rather than individual keys. The cnf claim binds each authorization token to the agent's current signing key while the identifier persists across key rotation. This reduces coupling between delegation history and key lifetime, but introduces lifecycle questions concerning delegation validity across key rotations—whether a delegation was issued before or after a rotation, and how invocations arriving after should be treated. Capability systems using delegation-specific or one-time keys largely avoid this issue, since delegated credentials typically expire before key rotation becomes relevant.

Its strengths lie in authenticated interoperability, cross-organizational execution, event-driven interaction, and maintaining coherent delegation history throughout mission execution. Unlike capability-based systems, authority remains associated with server-managed mission state rather than being embodied entirely within portable delegation artifacts.

This architectural choice intentionally separates authenticated execution from higher-level governance concerns. Authentication and delegation establish who is requesting execution and under what delegated authority. The Permission Endpoint now provides a normative execution-time decision surface: an agent may ask whether a specific action is permitted and receive granted or denied, recordable in the mission log. What remains outside the protocol's scope is the policy logic that produces that answer, together with attenuation policy, authority evolution, mission governance, and dynamic authority state.

Accordingly, AAuth is best understood as a server-mediated delegation and execution framework. It provides authenticated execution, delegation continuity, and mission coordination while allowing complementary policy and governance architectures to determine whether delegated authority remains valid as mission context, organizational policy, and authority state evolve.

Certificate Capabilities: zCaps and UCANs

Overview

Two of the specifications in this survey are implementations of the same underlying model, and this chapter evaluates them together. Authorization Capabilities ("zCaps", standardized in draft as Authorization Capabilities v0.3) and the User-Controlled Authorization Network ("UCANs") are both certificate capability systems: the authorization is a structured, cryptographically signed artifact that a delegator hands to a delegatee, carrying within itself everything a verifier needs to check it. Both descend from the object-capability tradition (UCAN cites SPKI as its direct ancestor), and both make the same core design choice: access is decided by possession ("do you hold a valid capability, and can you prove control of its key?") rather than by identity ("who are you, and what does my policy store say about you?").

The shared model works as follows. The root of authority is the resource owner; there is no central authorization server. A capability answers which agent can do what, with which resource, under which restrictions. Delegation is performed offline, by signing a new capability with a key the parent authorized; no round trip to the resource server or any other party is needed to delegate. Each delegation step must monotonically attenuate: a delegatee can pass on no more than they hold, along whatever axes the spec defines (actions, time, resource scope, policy). At invocation time, the delegatee presents the capability together with its full delegation chain and a proof of key possession, and the verifier checks the chain locally and cryptographically.

Because the two specs score nearly identically against the evaluation checklist, the scorecard below covers both in one table, with the shared reasoning stated once and per-spec nuances called out. The second half of the chapter covers where the specs genuinely diverge: encoding, resource scoping, revocation mechanics, chain representation, and a few features present in only one of the two.

Reference points used in this evaluation:

- the ZCAP v0.3 specification (the data model), and the zCap Developer's Guide, which documents the deployed state of the art; where deployment has diverged from the spec text (most notably: typed caveat objects are largely unused in production, replaced by URL-suffix attenuation, allowedAction, and expires), this is called out explicitly;
- the UCAN specification set: the overview plus the Delegation, Invocation, and Revocation sub-specs. (The Promise sub-spec is skipped as still marked draft.)

Scorecard

#	Requirement	zCaps	UCAN	Notes
1	Accountable (agent vs. principal/operator)	Partial	Partial	Same verdict for the same reason: every delegation edge is signed and thus attributable to a responsible delegator, but neither data model distinguishes "agent" from "legal-entity operator"; both are opaque DID-identified principals. Identity binding is delegated to a companion layer (VCs and issuer registries for zCaps; DID methods and higher-level identity systems for UCAN).

2	Resistant to confused deputy	Yes	Yes	By construction, as capability systems: designation and authorization travel together in the signed artifact (invocationTarget plus allowedAction; subject times command times policy), so a delegatee never exercises its own ambient authority on a resource designated by another party.

3	Represent authorization policies ("undelegatable", re-delegation rules, advisory "please")	Partial	Partial	Both provide an extensible policy slot and neither ships a standard vocabulary. zCaps reserve a generic caveat property, unused in deployment. UCAN defines a policy syntax (the pol property) and reserves the /ucan command namespace, but standardizes no caveats for delegation control. Neither has an advisory (non-enforced) tier; both are binary enforce/reject.

4	Chainable	Yes	Yes	The headline feature of the family. zCaps: parentCapability plus capabilityChain proof chains. UCAN: hash-linked self-certifying certificates rooted in the resource owner. Both cover operator to agent and agent to sub-agent, with links creatable offline.

5	Cross-organizational / locally verifiable	Yes	Yes	Verification is cryptographic and local; principals are DIDs, so no party needs an account on another's IAM system. zCaps mandate the full chain be carried in the invocation, with no verifier network access required. UCAN resolves proofs by CID from a local store, DHT, gossip, or REST. DID resolution and revocation freshness may still touch the network in both.

6	Attenuated	Yes	Yes	Both enforce monotonic attenuation, on different axes. zCaps: allowedAction subsets, no-later expires, URL path/query suffix narrowing of invocationTarget. UCAN: capability sets must not widen, nbf/exp validity intersection, command-lattice narrowing, policy statements.

7	Self-revocable (holder or anyone in chain)	Yes (at RS)	Yes (per Revocation spec)	Same guarantee, opposite mechanics. zCaps: any controller in the chain POSTs the zCap to an RS revocation endpoint; out-of-band, RS-enforced, online-only. UCAN: revocation is a first-class command (/ucan/revoke) with a Revoker role, expressed inside the same capability model; dissemination is left open (DHT, gossip, etc.).

+	Authentication / Proof of Possession	Yes	Yes	zCaps: HTTP Message Signatures (Cavage, migrating to RFC 9421) or a Data Integrity capabilityInvocation proof; the request itself is signed. UCAN: the Varsig envelope is signed by the issuer key, with a required nonce, time bounds, and signed invocations. Both are DPoP-analogous: an intercepted invocation cannot be replayed without the key.

+	Privacy of delegation chain	No	No	Both treat the chain as a provenance log, visible to the carrying agent and submitted whole to the verifier. No selective disclosure, encryption, or blinding in either. (zCaps: urn:uuid ids reduce correlation; ephemeral one-off keys approximate blinding. UCAN: explicitly offers no confinement.)
+	Offline-capable	Yes for delegation and verification	Yes for delegation and verification	The family's signature efficiency win: delegating requires no contact with the resource server. Chain verification is local. Invocation depends on reaching the resource; revocation depends on reaching the RS (zCaps) or on propagation (UCAN).

Detailed Evaluation

1. Accountable: agent vs. principal/operator

Both specs satisfy the accountability half of this criterion structurally and the differentiation half only implicitly.

The actor is named directly: the controller field of a zCap, or the issuer/audience/subject DIDs of a UCAN. Accountability across the chain is a structural guarantee in both. Every zCap delegation carries a capabilityDelegation proof signed by a controller of its parent, and every UCAN must be signed by the issuer DID's key, with the chain of tokens forming a provenance log of who delegated what to whom. For every edge in the delegation graph there is a cryptographically attributable, non-repudiable responsible party, and walking the chain back to the root recovers the complete provenance of any authority.

What neither spec provides is a semantic distinction between "this DID is an autonomous agent" and "this DID is the legal entity that stands behind it." A principal is an opaque DID; the root controller is the operator only by convention. Both specs push the binding of key identifiers to real-world legal entities explicitly out of scope, and for similar reasons: it is not always desirable or possible, and forcing it would be a privacy violation in many settings. Where the mapping is needed, both point at the same companion pattern: use an identity layer (Verifiable Credentials, issuer registries, DID methods) for the human-judgement call at the entry point, and use the capability chain for the flow of authority thereafter. An auditor can always identify which key delegated which authority to which other key; note that this auditability can be fulfilled with local pairwise identifiers only, without global identities.

2. Resistant to confused deputy

Resistant by construction, and this shared property is the reason the family exists. In both specs, designation and authorization are inseparable parts of the signed artifact: a zCap binds invocationTarget to allowedAction, and a UCAN capability is defined as subject times command times policy. The verifier evaluates the presented chain, not the invoker's own ambient standing, so there are no deputy permissions to confuse. A delegatee asked to act on a resource designated by someone else either holds a capability whose designation covers that resource, or the invocation fails.

3. Ability to represent authorization policies

The least emphasized area for both specs as currently deployed, and the clearest shared gap.

Both have the extension point. The zCap spec defines a generic caveat property: typed restriction objects (the spec's examples are ValidWhileTrue and DriveNoMoreThan) inherited down the chain, whose meaning "must be understood between the entity adding a caveat and the target evaluating it." UCAN goes further on machinery: policy is a dedicated top-level property (pol) with a defined syntax of statements and operators, alongside arguments (args) that constrain the command, and a reserved /ucan command namespace for community-blessed commands.

Neither ships vocabulary. In zCap deployments no caveat notation is currently used at all; attenuation is expressed implicitly through allowedAction, expires, and URL-suffix narrowing. UCAN's policy syntax expresses conditions on invocation arguments (the spec's examples are of the form "from this email address"), but standardizes nothing for delegation control.

Against the checklist's specific policy examples, the two specs fail identically:

- "Do not delegate further" / "this permission is undelegatable": not natively expressible in either. zCaps have no depth-limit or non-delegation flag; the only depth control is a verifier-global chain-length cap (the RS's blanket rule, not a per-capability instruction). UCAN likewise defines no such caveat; delegation is constrained only by the attenuation rules. (In the zCap design philosophy this is deliberate: forbidding delegation is considered an anti-pattern, since a delegatee can always proxy or share the controlling key.)
- Re-delegation policies: not standardized in either; both would need custom caveats or resource-specific policy semantics.
- Advisory "please" rules (desired re-delegation rules whose violation is not refused): no notion of advisory, non-enforced policy in either spec. Caveats and policies are evaluated at invocation and a failure invalidates the invocation. The model is binary enforce/reject, with no honor-if-possible tier.
- "Who gets to know what is a governance topic": consistent with the object-capability philosophy, both specs deliberately encode authority, not knowledge or identity policy; disclosure governance is out of scope.

So both data models can carry policy through an extensible slot, but neither ships a shared vocabulary for the policies the checklist cares about, and current deployments do not use the slots. This is arguably the highest-value area of future work for the family, if it is to serve agentic delegation.

4. Chainable

The headline strength of both specs and the reason the model exists.

In zCaps, delegation is expressed by parentCapability (pointing up at the parent or the root) plus a capabilityDelegation proof whose capabilityChain enumerates the ancestry. The root of authority is the resource itself: a root zCap controlled by the operator, from which authority flows root to agent to sub-agent. In UCAN, each token is a self-certifying certificate attesting a subset of its delegator's authority, hash-linked to its proofs and rooted in a resource owner; the Subject (sub) field marks whose authority the chain is about, and the chain is by definition a provenance log.

Both directly satisfy the two senses the checklist asks for: principal/operator to agent (root to first delegation), and agent to further subsystems or other agents (the zCap spec's valet example, car to Alyssa to Ben to Lem, is precisely a multi-hop agent-to-agent chain). In both, each link is created offline, simply by signing a new child artifact with a key the parent authorized; no involvement of the resource server is required to delegate.

5. Cross-organizational / locally verifiable

Strong in both, and again by design. Verification is purely cryptographic and local to the verifier: it traverses the chain from the root down, checking each delegation proof against the key authorized by the step above, and checking that each step only narrows authority. Principals are DID-based key identifiers, so two organizations can delegate to each other purely by exchanging signed capabilities, with no accounts on each other's IAM systems.

The specs differ in how the chain reaches the verifier. zCaps fix this normatively: a delegated zCap must carry its entire chain in the invocation, and a verifier must not be required to make network requests or database queries to dereference it; the root of trust is the RS, which dereferences the root zCap's controller from its own store. UCAN relies on content addressing: tokens are identified by CID, and proofs are resolvable from wherever the deployment keeps them (local store, DHT, gossip, REST), which makes chain transport a policy choice rather than a normative embedding rule.

Verification involves several steps, some of which (like DID resolution) may need to access the network, and some (like checking a local revocation registry) can be done offline. To put it another way, chain structure verification can be performed offline; key revocation checks and other verification steps may need network access.

6. Attenuated

Both specs enforce monotonic attenuation, checked by the verifier at every link: a delegator can hand out no more than they hold. The axes differ.

zCaps attenuate along three deployed axes: allowedAction in a child must not be less restrictive than the parent's; expires must not be later than the parent's (and every delegated zCap must have an expiry); and invocationTarget may be narrowed by appending a URL path or query suffix (/bars/123 to /bars/123/bazzes/456?day=tuesday). The caveat property is reserved as a fourth axis but unused in practice.

UCAN attenuates abstractly: each direct delegation must restate or diminish its capabilities, the validity interval is the intersection across the chain (latest nbf to earliest exp), and commands narrow along a lattice, where shorter command paths prove longer ones in the same namespace (/crud can prove /crud/read but not /stack/pop). Policy statements and arguments provide further attenuation axes.

The limitations are mirror images. zCap attenuation of the resource is restricted to URL-suffix shaping, fine for hierarchical REST resources but awkward for non-hierarchical scoping, and a single zCap carries exactly one invocationTarget (multiple target/action combinations are listed as future work). UCAN's abstraction is more flexible but less prescriptive: resource scoping is left to how cmd and args are interpreted, with no normative URL rules in the core spec.

7. Self-revocable

Both specs meet the checklist requirement that the holder, or someone in the delegation chain, be able to revoke (modulo network partitions), and this is the criterion where their mechanics diverge most sharply.

zCaps treat revocation as a resource-server state change. Any controller appearing in a delegated zCap's chain may revoke it (revoking everything downstream) by POSTing the zCap to the RS's revocation endpoint; the spec sketches a /zcaps/revocations subpath whose root zCap is controlled by all controllers in the chain, precisely so that any ancestor can revoke. Mandatory expires on every delegated zCap bounds how long a revocation must be remembered and bounds damage from a lost-but-unrevoked capability. The mechanism is out-of-band and RS-enforced: online-only, with the RS as the single enforcement point.

UCAN treats revocation as a first-class operation inside the same capability model. The Revocation sub-spec defines a Revoker role (an issuer listed in a proof chain who revokes a UCAN) and revocation is expressed as a command in the reserved /ucan namespace; executors must validate revocation state at execution time. How revocation notices propagate is left to the deployment (DHT, gossip, and similar patterns are all permitted), so there is no mandated always-online central point, at the cost of unspecified dissemination guarantees.

The shared characteristic worth stating plainly for agentic use cases: in neither spec is revocation embedded cryptographically in the token itself. It is state that the enforcement point (the RS, or the executor's revocation store) must learn about, so offline revocation semantics are unavailable in both.

Nice-to-haves

Authentication / Proof of Possession

Possession alone is never sufficient in either spec; invocation requires proving control of the authorized key.

zCaps define two binding mechanisms: an HTTP Message Signature (deployments use Cavage Draft 12 today, migrating to RFC 9421) over a Capability-Invocation header, plus Digest/Content-Digest for bodies; or a Data Integrity capabilityInvocation proof attached to a request document. Either binding mechanism requires a digital signature. Note that as with all proof-of-possession signature mechanisms, additional replay attack mechanisms are required (and used by zCaps), such as short expiration periods and unique ids tracked by the resource server/verifier.

UCAN builds proof of possession into its envelope: every token is signed by the issuer DID's key over the canonical DAG-CBOR payload, and replay protection is layered (a required nonce, nbf/exp time bounds, and signed invocations at the transport layer). The intent is the same as zCaps' HTTP signatures; the expression lives in the UCAN envelope rather than in a separate signing convention.

Privacy of the delegation chain

No, in both. The full chain must reach the verifier on invocation and is carried by (and therefore visible to) the invoking agent itself; the principal, the verifier, and the token carrier all see the whole chain. Neither spec defines selective disclosure, encryption, or blinding of intermediate delegatees. UCAN is explicit that chains are provenance logs and that UCANs do not offer confinement; delegatees can sub-delegate without alerting their delegator. zCaps use urn:uuid identifiers to reduce correlation, and most of the desired effect can be approximated with ephemeral one-off keys, but neither is a chain-privacy mechanism. In both systems, additional information associated with the actors in the chain, if any, is stored usually out-of-bound as non-key DID document contents, verifiable credentials, or other identifier-anchored metadata.

Offline-capable

The offline story is the same for both, split by lifecycle stage, and it is the family's signature efficiency advantage:

- Delegation: fully offline. Delegating is signing a new child artifact with an authorized key; no contact with the resource server or any authorization server.
- Chain verification: offline-capable. The verifier checks the supplied or resolved chain locally (subject to the DID-method and revocation-freshness caveats noted under #5).
- Invocation: depends on reaching the resource. If the RS or executor is reachable offline (local machine, LAN), invocation can be offline too.
- Revocation: depends on reaching the RS (zCaps) or on the deployment's propagation pattern (UCAN), per #7.

Where the Two Specs Diverge

The evaluation above shows the two specs scoring as a family. This section highlights features that exist in one spec but not the other. (The revocation divergence, endpoint versus command, is covered under criterion 7 above.)

Multiple actions per capability

A delegated zCap may carry an allowedAction that is a single string or an array of strings (for example "allowedAction": ["write", "read"]), so one capability can directly encode "read and write but not delete", with verifiers enforcing that a child's array is no less restrictive than its parent's. A UCAN payload has exactly one cmd string; the spec defines no list form. To grant multiple methods, UCAN relies on either a more general command (such as /crud) whose policy and arguments restrict which operations are allowed, or multiple UCANs, each with its own cmd.

Asymmetry: zCaps have built-in multi-action support; UCAN pushes multi-method expressiveness into policy or multiple tokens.

Resource scoping: URL attenuation vs. command lattice

zCaps make URL structure part of the attenuation rule: a child's invocationTarget must equal the parent's or extend it with a path or query suffix, with normative rules for the suffix forms. UCAN has no invocationTarget equivalent; resources are governed by the Owner/Subject relationship and command semantics, and narrowing happens along the command lattice (segment structure matters: shorter commands prove longer paths in the same namespace) plus arguments, with no normative URL rules.

Asymmetry: zCaps hard-code hierarchical URL narrowing in the data model; UCAN scopes resources through its command and argument structure.

Capability composition and authority union

UCAN defines a spec-level algebra for combining authority: the set of capabilities delegated by a UCAN is its "authority", merging capability authorities must follow set semantics (the result includes all capabilities from the inputs), and authorities can be combined in any order. This supports compositionality across resources, with no distinction between co-located and distributed resources. zCaps define only single-root, single-target chains: one parentCapability per delegated zCap, one invocationTarget per capability, and no defined algebra for merging multiple chains into a combined authority.

Asymmetry: UCAN specifies authority union; zCaps do not. (As discussed in the evaluation checklist, composability is trivially satisfied by any capability system at the model level; the asymmetry here is about spec-level machinery, not achievability.)

Container and signature format

zCaps are JSON-LD documents (with a zcap context plus a security suite context) signed with Data Integrity proofs; there is no content-addressing requirement. UCANs are DAG-CBOR payloads inside a Varsig envelope, identified by CIDv1: native IPLD objects, with the signature scheme described by the Varsig header. This is the divergence that most shapes each spec's natural habitat: zCaps sit on the web-standards stack (JSON-LD, Data Integrity, HTTP signatures), UCAN on the content-addressed, local-first stack (IPLD, CIDs, replicated stores).

Chain structure and storage

zCaps fix the wire representation of the chain: the capabilityChain must include the root zCap's id and every ancestor by id, except the immediate parent, which must be fully embedded; verifiers must not be required to perform network requests to dereference the chain. UCAN defines chains logically and relies on content-addressed storage and transport: proofs may be carried inline or referenced by CID and resolved from a delegation store, DHT, gossip, or REST endpoint, with no prescribed embedding pattern.

Asymmetry: zCaps trade transport size for guaranteed verifier self-sufficiency; UCAN trades a resolution step for flexible, deduplicated proof storage.

Invocation mechanism

zCaps define invocation conventions for two carriers: an HTTP capability-invocation header (with the zCap attached compressed in a header parameter) signed with an HTTP signature, or a Data Integrity capabilityInvocation proof on an arbitrary JSON-LD document. UCAN reuses its own envelope: the Invocation sub-spec describes an Invoker sending UCANs plus args to an Executor, which validates signatures, chains, nonce, time bounds, and revocation; transport is left open, and Data Integrity is not used.

Powerline (UCAN only)

UCAN's delegation work defines Powerline, a pattern for automatically delegating all future delegations to another agent regardless of Subject. Once established, it acts as a standing delegation conduit: whenever any issuer delegates along that line, authority transparently flows to the designated sink agent without being explicitly named in each delegation. It is built from normal UCAN pieces (principals, subjects, delegation rules) rather than a new primitive, and it is useful for centralizing audit, enforcement, or mediation in a specific agent, for example a storage or orchestration service that should automatically receive all capabilities granted in a namespace.

Asymmetry: zCaps have no analog to the Powerline pattern.

Summary

zCaps and UCAN are two realizations of the same certificate-capability model, and against the evaluation checklist they score as a family: strong on the mechanics of delegated authority (chaining, monotonic attenuation, locally and cross-organizationally verifiable proof chains, proof of possession, offline delegation), and weak in the same two places. Neither distinguishes agent from operator in its data model (both delegate identity binding to a companion layer such as VCs), and neither ships a standard policy vocabulary for delegation control ("undelegatable", re-delegation rules, advisory semantics), despite both reserving the extension point for one. Chain privacy is absent from both, and revocation in both is state the enforcement point must learn, not a property of the token.

Those shared gaps are family-level gaps, which makes them natural working-group targets: a standardized caveat/policy vocabulary and a companion identity-binding mechanism would lift both specs at once.

The differences between the two are real but sit a level down, in encoding and ecosystem rather than capability model: JSON-LD and Data Integrity on the web-standards stack versus DAG-CBOR and CIDs on the content-addressed local-first stack; normative chain embedding

versus resolvable proof stores; multi-action arrays versus a command lattice; an RS revocation endpoint versus a revocation command. For delegated authorization for AI agents, choosing between them is largely a deployment-ecosystem decision; the properties that matter for the agentic checklist, and the work still needed, are common to both.

The Cedar Policy Language

Overview

Cedar is an open-source authorization policy language and evaluation engine created by IAM specialists at Amazon with formal methods training (used in AWS Verified Permissions; the language syntax itself is now a CNCF candidate project). A Cedar policy answers "**may** this principal **perform** this action **on** this resource **under** these conditions (context)" using `permit` and `forbid` statements validated against a typed schema. Evaluation is deterministic, default-deny, and `forbid` always overrides `permit`. Over time, the core Cedar language and its canonical reference implementation in Rust have been generalized from Amazon's production implementation in Keycloak/AWS contexts to be hardened and governed at CNCF within the Linux Foundation (as mentioned above); a cloud-agnostic IAM framework generalizing the Keycloak/AWS tooling is governed in a dedicated Linux Foundation Project called Janssen, which also governs and develops the younger "Cedarling," a lightweight version of Cedar's enforcement engine that can be run "practically anywhere" as a WASM component inside of not just servers but even clients and agents/harnesses.

//TODO: add section explaining that machine-readable policy evaluation CAN be construed to include identity-blind/unauthenticated capabilities, OR AT LEAST to help such tooling be intelligible to agents and unfamiliar callers in a cross-org use-case.

Cedar has no native, enforced delegation primitive. It is a policy expression language plus a Policy Decision Point (PDP), with no native concept of a credential, a holder, a signature, or a delegation chain. It is *identity-based* in the classic sense: the verifier decides access by evaluating who the principal is (and their attributes and group memberships, as made known to the verifier by tokens from server-side tooling like Janssen) against policies the verifier holds, in contrast to possession-based models like zCaps, where presenting a valid signed capability constitutes the "just-in-time" proof of authority. This identity-vs-possession distinction drives nearly every row of the scorecard.

Cedar is therefore evaluated here **as a component in a delegation stack**: Cedar supplies policy expression and local evaluation in cross-organization use cases where policy must be made intelligible to callers and counterparties across policy boundaries; the delegation mechanics (chain construction, portable proof, holder-side revocation) must be supplied by a carrier layer around it, but the policy anchoring the semantics of those delegations and error messages can be more meaningfully communicated by the combination of the layers.

Three reference points are used:

- the Cedar language and documentation (the "spec": syntax, schema, validation, templates, evaluation semantics),
- the Janssen Project Cedarling, the most relevant deployed embedding: a lightweight, embeddable PDP (Rust core with WASM, mobile, Python and Java bindings) that implements Token-Based Access Control (TBAC), deriving Cedar principal entities from JWTs issued by trusted issuers and evaluating policies locally at the edge, and
- Clawdrey Hepburn, a first-person agentic case study: an autonomous AI agent whose operator-defined action boundaries (which accounts it may follow, what it may do with its own credentials and payment card) are enforced by an embedded Cedar engine checked before every action, with a typed schema modeling the agent, its operator, and its action space.

Where Cedar (the language) and runtime-evaluation deployments like Cedarling differ in what they provide, this is called out, since several matrix rows are satisfiable only by the deployment layer.

Scorecard

#	Requirement	Verdict	Notes

1	**Accountable** (agent vs. principal/operator)	Yes (at the modeling level)	The schema can type the distinction directly (e.g. `entity Agent in [User]`); Cedarling derives separate User and Workload principals from the `id_token` and `access_token` and can require both to be authorized. But the distinction is data the verifier trusts an Authorization Server and token exchange to provide, not a cryptographically attributable delegation edge.
2	**Resistant to confused deputy**	No	Identity-based by design: the deputy exercises its own standing permissions against a caller-designated resource, and the model does not combine designation with authorization. Policies can condition on request context to mitigate specific deputy scenarios, but that is mitigation, not structural immunity.

3	**Represent authorization policies** (embed/link; "undelegatable", re-delegation rules, advisory "please")	Yes, except advisory	This is Cedar's core competence. "Do not delegate further" and re-delegation rules are expressible if delegation itself is modeled as an action. No advisory tier: decisions are binary permit/forbid, with no obligations or native "honor-if-possible" semantics.
4	**Chainable**	No (stack-dependent)	No delegation primitive, no chain artifact. Delegation can be modeled as verifier-side **state** (entities or instantiated policy templates), but chains are not portable, signed, or first-class. In a delegation stack, the chain must live in the carrier or elsewhere in the Authorization stack; Cedar evaluates each hop atomically.

5	**Cross-organizational / locally verifiable**	Partial	Evaluation is fully local (Cedarling runs in browsers, mobile, gateways; multi-issuer JWTs supported without shared IAM accounts). But the policy store is unilateral, that is -- the verifier's. There is no standard or guidance for exchanging ~~or federating~~ policy semantics across organizations in an adhoc way, and trusted issuers must be pre-configured~~.~~, BUT core contributors do contribute to and recommend using AuthZEN for federation and runtime issuer-list
6	**Attenuated**	Partial	Forbid-overrides-permit gives monotonic restriction: layering additional forbid policies (or a stricter verifier-side policy set) can only shrink authority, never expand it. But Cedar has no runtime check that a delegatee's grant is a subset of the delegator's; subset/equivalence can be proven offline with Cedar's symbolic analysis tooling. Attenuation enforcement is a stack responsibility.

7	**Self-revocable** (holder or somebody in chain)	Partial	Revocation is a policy store change (remove a permit or add a forbid), immediate at the next evaluation and very fine-grained. But it is performed by whoever administers the policy store, not natively by a delegator or delegatee in the chain; mapping chain participants to revocation rights is a stack design task. Cascade to downstream grants is not native either: it holds only if delegation is modeled as linked entities whose policy re-checks the ancestry.
+	Authentication / proof of possession	No (out of scope)	Cedar does not authenticate. Cedarling validates JWT signatures (proof of issuance by a trusted issuer), which is not holder proof of possession; PoP binding is left to the token layer.

+	Privacy of delegation chain	Possible (by architecture)	There is no chain in the data model, so nothing is carried in the request: a delegatee presents only their own token/identity. Intermediate delegatees need not learn the rest of the chain. The flip side: the verifier's policy store and entity data must hold whatever graph exists, so the verifier sees everything.
+	Offline-capable	Partial	Evaluation is fully offline once the policy store is loaded (this is Cedarling's headline feature). But *granting* a delegation means writing to a policy store, which is an online, administrative act, the opposite of zCaps' offline delegation-by-signing.

(Cedar) detailed evaluation

1 - Accountable, agent vs. principal/operator

Differentiation is a schema design choice, made trivially. Cedar entities are typed and hierarchical, so the agent/operator relationship can be modeled directly, for example `entity Agent in [User] { model: String, runtime: String, trust_level: String }`, which is exactly the pattern the Clawdrey case study uses: the agent is a typed principal nested under its human operator, and policies can condition on either or both. Cedarling

operationalizes the same split out of the box: it constructs a **User** principal from the OIDC id_token and a separate **Workload** principal from the OAuth access token (the software acting on behalf of the person, or autonomously), and its default decision logic can require that *both* the person and the workload be authorized for the request to proceed. That is precisely the agent-vs-operator distinction the matrix asks for, expressed natively in the data model rather than by convention.

What Cedar does not provide is cryptographic accountability for the *delegation* itself. There is no signed delegation edge: the "fact" that operator O stands behind agent A is an entity attribute or a token claim that the verifier chose to trust. Attribution quality is therefore exactly the attribution quality of the surrounding identity layer (the JWT issuer, in Cedarling's case). An auditor can read the decision logs (Cedarling logs every decision with diagnostics) and reconstruct *which policy permitted which principal*, but cannot derive a non-repudiable "this delegator authorized this delegatee" proof from the Cedar artifacts alone. The accountable party for any grant is, structurally, the policy store administrator.

2 - Resistant to confused deputy

Evaluated as-is: no, and the exposure follows directly from the identity-based model. The verifier decides access by evaluating who the principal is against verifier-held policies, so a deputy service acting on a caller-designated resource exercises its own standing permissions; nothing in the model binds the caller's designation to the authority being used. Policies conditioned on request context can mitigate specific deputy scenarios, but that is per-policy mitigation, not the structural designation-plus-authorization coupling that capability systems provide. In a delegation stack, immunity would have to come from the carrier layer, with Cedar evaluating the rules at each hop.

3 - Ability to represent authorization policies

Policies are first-class, schema-validated, and arbitrarily conditional on principal, action, resource and request context: attribute comparisons, group membership (`in`), set operations, string and IP functions, and explicit `forbid` statements that override any permit. Policies can be embedded with the verifier or distributed to it (Cedarling fetches signed policy stores over HTTPS or from a Lock Server), satisfying the embed-or-link condition.

Against the matrix's specific examples:

- **"Do not delegate further" / "this permission is undelegatable":** expressible, with one precondition: delegation must be modeled as an action in the schema (e.g. `action`

`delegate appliesTo { principal: [Agent], resource: [Permission] }`). Then `forbid (principal, action == Action::"delegate", resource == Permission::"x");` makes permission x undelegatable, and since forbid overrides permit, no later grant can reopen it. Depth limits are expressible the same way if depth is carried as an attribute. The precondition matters: Cedar can only forbid delegations that the stack routes through a Cedar check.

- **Re-delegation policies:** same mechanism, with the full expressiveness of the language (delegate only to principals in a given group, only with attenuated scopes, only before an expiry, and so on).
- **Advisory "please" rules (a request carrying desired re-delegation rules whose violation is not refused):** not supported. Cedar evaluation returns a binary Allow/Deny plus diagnostics; there is no obligations or advice tier in the decision (a notable contrast with XACML), and policy annotations like `@id(...)` are non-semantic metadata. An advisory tier would have to be built beside Cedar, not in it.
- **"Who gets to know what is a governance topic":** Cedar is agnostic; it evaluates whatever entity data it is handed. Disclosure governance sits in the surrounding stack (which claims go into tokens, which attributes into the policy store).

One genuinely distinctive property deserves mention here: Cedar is **analyzable**. The language was deliberately kept non-Turing-complete (no loops, no recursion, guaranteed termination) and its evaluator is formally verified, so policy sets can be subjected to automated reasoning: does policy set A permit anything B does not, are these two sets equivalent, is this forbid ever reachable. For a delegation context, that means delegation rules can be *audited and proven* offline, which no other technology in this survey offers.

4 - Chainable

Evaluated as-is: no. Cedar has no delegation primitive. There is no parent pointer, no chain, no signed artifact that one party hands to another. Two in-language mechanisms get part of the way and are worth naming precisely:

- **Policy templates.** A template with `?principal` / `?resource` slots can be instantiated at runtime to grant a specific principal access to a specific resource. Template instantiation *is* a grant operation and can be wired up as the delegation act ("delegator triggers instantiation of a template scoped to the delegatee"). But instantiation is an administrative write to the policy store, performed by whoever holds write access, and the resulting policy records no ancestry.
- **Delegation modeled as data.** The schema can define delegation entities (delegator, delegatee, scope, expiry, parent reference) and policies that grant access when a valid delegation entity exists. This can represent chains of arbitrary depth, including agent-to-subagent links. But the chain is then application state in the verifier's entity

store: not portable, not signed, not independently verifiable, and entirely a bespoke convention.

So in a delegation stack, the chain must live in the carrier layer (signed tokens or credentials linked hop to hop), and Cedar's role is to evaluate the rules *at* each hop: may this delegator delegate this, may this delegatee exercise it. Cedar is the rulebook for the chain, not the chain.

5 - Cross-organizational / locally verifiable

Half of this requirement is Cedarling's headline feature; the other half is structurally missing.

Locally verifiable: yes. Cedar evaluation requires no network call: policies, schema and entity data are in memory, and decisions are sub-millisecond (for simple policies only). Cedarling pushes this to the edge deliberately: a WASM/mobile/server-embeddable PDP that loads a policy store at startup and answers authorization questions locally, so an API gateway, browser or device enforces policy without phoning a central authorization server per request. JWT signature validation against trusted issuers is also local (cached JWKS).

Cross-organizational: partial. Cedarling's TBAC model is genuinely multi-party at the *evidence* level: `authorize_multi_issuer` accepts JWTs from several different trusted issuers in one request and maps their claims to Cedar entities, so organization A's verifier can consume identity evidence issued by organization B without B having an account in A's IAM. But the *policy* level is unilateral: the policy store (schema, policies, trusted issuer list) belongs to the verifier alone. There is no standard for one organization to hand another a policy fragment with agreed semantics, no shared schema federation, and the trusted issuer list is pre-configuration, not dynamic trust establishment. Two organizations can interoperate, but only after bilateral setup; the matrix's "express authorization policies without accounts on each other's IAM systems" is met for authentication evidence and unmet for policy exchange.

6 - Attenuated

Cedar has one structural property that maps beautifully onto attenuation, and one missing property that prevents a full Yes.

The structural property is **monotonic restriction by composition**: `forbid` overrides `permit`, and evaluation is over the union of all policies. Adding policies to a set can therefore only ever *shrink* what a forbid touches and a verifier (or any later layer) can stack stricter policy on top of a base grant with a guarantee that the stack cannot widen authority. This is a natural fit for the delegation-stack pattern in which each hop attaches further restrictions: express the delegator's

grant as the base policy set and each subsequent restriction as additional forbids or narrower permits evaluated together.

What is missing is **enforced subset semantics between grants**. If a delegation is represented as "delegatee gets policy set B, derived from delegator's policy set A", nothing in the Cedar runtime checks B ⊆ A; an over-broad B is just another valid policy set. The check is possible, and this is where Cedar is unique: its analyzability means "B permits nothing A does not" is a decidable question answerable by symbolic analysis tooling at delegation time or audit time. But it is an offline/toolchain guarantee the stack must invoke, not a property the evaluator enforces per request the way a zCaps verifier enforces allowedAction narrowing.

Expressiveness of attenuation, once enforced, is far richer than URL-suffix narrowing: any condition the language can state (attribute ranges, context conditions, time windows, resource groups) can be the attenuation axis.

7 - Self-revocable

Revocation in Cedar terms is straightforward and immediate: delete the permit (or the template instantiation, or the delegation entity) or add a forbid, and the very next evaluation reflects it. Forbid-overrides-permit makes "revoke" particularly clean: a targeted forbid kills a grant without having to find and remove every permit that contributed to it. Granularity is excellent: one delegatee, one action, one resource, one condition.

The matrix question, however, is *who* can revoke. Natively: whoever can write to the policy store. That is an administrative role, not a chain position. A delegator can revoke what they delegated only if the stack maps "delegator of grant X" to "may modify/forbid grant X in the store", which Cedar can itself express as policy (delegation management actions authorized by Cedar policies, the recursive pattern) but does not provide out of the box. Propagation is also a deployment property: Cedarling instances cache their policy store and pick up changes on refresh or via Lock Server push, so revocation latency equals policy distribution latency. Within a single trust domain this is near-immediate; across many edge PDPs it is eventually consistent.

On **cascade** (does revoking a grant revoke everything delegated below it): Cedar has no built-in notion of a chain, so there is no built-in cascade either. Whether revocation propagates downstream depends on how delegation is modeled. If each grant is an independent policy, revoking Alice-to-Bob leaves Bob-to-Carol untouched, and Carol keeps her access until that grant is separately removed. Cascade is achievable by modeling delegation as data with parent links plus a permit policy that succeeds only when the whole ancestry is still valid: each

delegation is an entity carrying a parent reference, and Carol's evaluation walks up the chain, so revoking Alice-to-Bob cascades to Bob-to-Carol because the walk hits the now-missing or now-forbidden link and fails closed. This is a contrast with zCaps, where cascade is structural (a delegated capability embeds its parent, so rejecting any link fails everything below it by construction); in Cedar cascade is a property you build into the policy and entity model, not one the engine provides.

Net: revocation is a strength of the centralized-policy model (no token to chase, no expiry to wait out, unlike credential-based systems where an issued artifact is "out there"), but holder/chain self-revocation is a stack construct, not a Cedar feature.

Composability (spec-level note)

A clear yes. Cedar evaluates every request against the union of all loaded policies; an Allow needs at least one satisfied permit and no satisfied forbid, and the satisfied permit(s) can originate from any source loaded into the store. Principals can belong to arbitrarily many groups and roles simultaneously (the matrix's RBAC single-role counter-example does not apply: Cedar's `in` hierarchy is a DAG, not a single assignment). Cedarling extends composition to the evidence layer: `authorize_multi_issuer` lets one request carry tokens from multiple independent issuers, each mapped to entities, so authority deriving from several principals or organizations is combined in a single decision.

The one caveat mirrors row 5: all the composed sources must already be present in (or mapped into) the verifier's policy store and trusted issuer configuration by stable identifiers. Composition is native at decision time, but onboarding a new source of authority is configuration.

Nice-to-haves

Authentication / proof of possession

Out of scope for Cedar by design: the engine assumes the principal has already been authenticated and evaluates the request it is handed. Cedarling adds issuance-side verification: JWT signature validation against pre-configured trusted issuers, aud/sub cross-checks between tokens, optional status checks, and exp/nbf checks. This proves the tokens were issued by a trusted party and are current, not that the presenter is the rightful holder. Sender-constrained tokens (DPoP/mTLS-bound) can be layered in by the token infrastructure, but Cedar/Cedarling does not itself verify possession.

Privacy of the delegation chain

An interesting inversion of the credential-based technologies. Because nothing is carried in the request, the matrix's desired property (the intermediate delegatee does not learn the rest of the chain) falls out of the architecture almost for free: each delegatee authenticates as themselves and presents their own tokens; whatever delegation graph exists is reconstructed verifier-side from the policy store and entity data. The principal (as policy author) and the verifier can know the full story while the token carrier knows only their own grant.

The cost is the mirror image: the verifier's policy store must contain the graph, so the verifier (and the policy store infrastructure) sees everything, and chain privacy *from the verifier* is impossible. Whether this counts as satisfying the row depends on which party the subgroup is trying to blind; for the stated formulation (hide the chain from the intermediate agent, principal and verifier know all) Cedar-style architectures actually do well.

Offline-capable

- **Evaluation/use: fully offline.** This is Cedarling's reason for existing: the PDP is embedded (browser WASM, mobile, gateway, local process) and decides locally with no network round trip. Local-machine and LAN-only agentic use cases are well served; the Clawdrey deployment is exactly this shape (an agent on a single machine consulting its local Cedar engine before each action).
- **Delegation/granting: online and administrative.** Creating a new grant means writing to a policy store and distributing it. There is no offline delegate-by-signing; the efficiency advantage the matrix highlights (delegate without contacting anyone) is absent.
- **Revocation: online to the store, then eventually consistent** to edge PDPs on refresh.

Summary

Cedar is, almost row by row, the complement of the capability-based technologies in this survey. It is strongest exactly where zCaps are weakest: rich, analyzable, schema-validated policy expression (including "undelegatable" and re-delegation rules), native agent-vs-operator modeling, composition of authority from multiple sources in one decision, fine-grained immediate revocation, and fully local evaluation at the edge. It is weakest exactly where zCaps are strongest: there is no delegation chain, no portable signed grant, no holder-side revocation, no offline delegation, and no cryptographic attribution of who authorized whom; all of these are verifier-side state administered through a policy store.

For delegated authorization for AI agents, the implication is that Cedar is not a candidate *delegation data model* but a strong candidate *policy layer inside one*: a carrier technology supplies the chain, proof and portability, while Cedar supplies the rulebook each hop is checked against, with the unusual bonus that those rules can be formally analyzed (delegation

attenuation provable offline). The Cedarling deployment shows the integration pattern already in production form (policies travel to the edge, identity evidence travels as multi-issuer JWTs, decisions stay local), and the Clawdrey case study shows the agentic pattern (operator-authored Cedar policies as the hard boundary on an autonomous agent's action space). What no current Cedar deployment shows is the missing piece the matrix cares most about: a standardized way for a delegator to hand a delegatee a portable, attenuated, chain-verifiable grant.

References

Cedar language and engine

- Cedar project site — positioning, project overview, CNCF status.
- Cedar documentation — primary language reference.
  - Basic Cedar syntax — permit/forbid effects, scope, when/unless conditions, and the forbid-overrides-permit / default-deny combining rules.
  - Terms & concepts — entities, typed UIDs, the entity hierarchy, static vs. template-linked policies, RBAC/ABAC/ReBAC framing.
  - Policy templates and Roles with policy templates — `?principal` / `?resource` slots, template-linked policies as a runtime grant mechanism, and their coupling to user life-cycle management (the "delegation as template instantiation" argument).
- Cedar: A New Language for Expressive, Fast, Safe, and Analyzable Authorization (extended version, arXiv 2403.04651) — the design paper; basis for the non-Turing-complete / formally-verified / analyzable claims and the two-slot template restriction.
- cedar-policy/cedar (GitHub) and cedar_policy crate (docs.rs) — reference implementation; entity store model, local in-process evaluation, validator/typechecking.

Janssen Project Cedarling (the deployed embedding)

- Cedarling overview / getting started — embeddable PDP on the Rust Cedar engine, WASM/mobile/server bindings, Token-Based Access Control, local edge evaluation.
- Authorization using Cedarling — derivation of separate User (id_token) and Workload (access_token) principals, the "person AND workload must be authorized" decision logic, JWT validation steps.
- Cedarling getting started — multi-issuer authorization — `authorize_multi_issuer` and `authorize_unsigned` methods; multiple trusted issuers in one request.
- Policy Store — schema + policies + trusted issuers bundle, claim mapping, role mapping; loaded as static JSON or fetched over HTTPS / from a Lock Server.

Agentic case study

- Clawdrey Hepburn — autonomous AI agent using an embedded Cedar engine as its action-space boundary.
    - Why I Built a Policy Engine Before I Built a Personality — per-action Cedar checks as enforced (not advisory) boundaries; the "follow only accounts Sarah follows" example.
    - Modeling an Agent's World in Cedar — typed schema modeling the agent, operator, and action space; agent nested as a principal under its human operator.

AuthZEN

Overview

This chapter evaluates AuthZEN against the seven mandatory criteria and the three additional considerations defined in the Overview chapter.

For the purposes of this assessment, we distinguish between the AuthZEN Core Specification and the broader AuthZEN ecosystem.

The term Core Specification refers to the AuthZEN Authorization API 1.0 (hereafter referred to as "Core"), which defines the normative interoperability model between a Policy Enforcement Point (PEP) and a Policy Decision Point (PDP). As the primary standards-track specification and the foundation for runtime authorization interoperability, it serves as the baseline for evaluating AuthZEN against each criterion.

The broader AuthZEN ecosystem currently includes three complementary specifications:

It is important to note that these specifications are not all at the same level of maturity. AuthZEN Authorization API 1.0 is a Final Specification and serves as the normative foundation for this assessment. By contrast, COAZ and COAZ-MCP are currently Draft 1 specifications, while the AuthZEN Policy Store Format remains a working draft (GitHub Repo). Accordingly, any assessment of the broader AuthZEN ecosystem should be understood as including current draft specifications in addition to the finalized Core specification.

- AuthZEN Policy Store Format (hereafter referred to as "Policy Store"), which defines a portable and PDP-neutral packaging format for policy artifacts, schemas, trusted issuers, metadata, and related configuration.
- COAZ: A Framework for Mapping Information Models to AuthZEN Authorization Requests (hereafter referred to as "COAZ"), which defines a protocol-neutral framework for translating external protocol interactions into AuthZEN Subject-Action-Resource-Context (SARC) authorization requests.
- COAZ-MCP: COAZ Binding for the Model Context Protocol (hereafter referred to as "COAZ-MCP"), which applies the COAZ model specifically to the Model Context Protocol

(MCP), enabling parameter-aware authorization decisions for AI agents, MCP gateways, and MCP servers.

It is important to note that these specifications are not all at the same level of maturity. AuthZEN Authorization API 1.0 is a Final Specification and serves as the normative foundation for this assessment. By contrast, COAZ and COAZ-MCP are currently Draft 1 specifications, while the AuthZEN Policy Store Format remains a working draft (GitHub Repo). Accordingly, any assessment of the broader AuthZEN ecosystem should be understood as including current draft specifications in addition to the finalized Core specification.

The evaluation therefore follows a layered approach. Each criterion is first assessed against the capabilities provided by the Core specification alone. Where relevant, the analysis then identifies how Policy Store, COAZ, or COAZ-MCP specifications materially strengthen, clarify, extend, or operationalize those capabilities. This distinction prevents ecosystem features from being incorrectly attributed to the core protocol while still allowing AuthZEN to be evaluated as a practical authorization framework for agentic systems.

Its central data model is a runtime authorization request expressed in terms of Subject, Action, Resource, and Context (SARC). The core specification defines single Access Evaluation and multiple Access Evaluations endpoints together with Search operations for discovering permissible resources, subjects, or actions. At the same time, it explicitly excludes policy language, architecture, policy state management aspects of a PDP and API authentication from its scope. Accordingly, this chapter distinguishes functionalities that are intrinsic to the AuthZEN protocol from functionalities supplied by the PDP, policy engine, identity system, or another surrounding layer.

The Policy Store addresses the policy-management side of the architecture by defining a PDP-neutral package, the bundle of policy files in a directory structure, for co-locating policies, schema, default entities, trusted token issuers, and supporting metadata. Also, this specification defines Policy Store API to manage the package. Its purpose is portability, versioning, auditability, and interoperability of policy artifacts across PDPs and tooling. Importantly, it standardizes the packaging and management format rather than the underlying semantics of a particular policy language or engine, thereby preserving implementation flexibility within PDPs while enabling interoperable policy distribution and lifecycle management.

COAZ addresses the boundary between external protocols and AuthZEN. It defines a protocol-neutral framework for declaratively mapping the inputs of an incoming operation into a SARC authorization request. Values may be fixed or computed from operation inputs, normally using Common Expression Language (CEL), allowing authorization decisions to reflect the actual parameters and context of an operation rather than only a caller identity. COAZ itself is protocol-agnostic, while protocol-specific bindings such as COAZ-MCP define how a particular protocol is projected into the SARC model.

Finally, COAZ-MCP applies that framework specifically to the Model Context Protocol. It maps MCP JSON-RPC operations into AuthZEN evaluations, providing default mappings for MCP methods and allowing tool-specific mappings where greater precision is required. This is particularly relevant to AI-agent authorization because it enables fine-grained, parameter-level policy evaluation at MCP gateways and servers. In the same way, future bindings could apply COAZ to other protocols, such as HTTP APIs, OpenAPI-described routes, gRPC services, or additional agent frameworks.

Many of the entries in the scorecard are marked as No, but this does not imply that AuthZEN is poorly suited to agentic authorization. Rather, they reflect the fact that AuthZEN addresses a different layer of the problem. Its primary role is to standardize runtime access decisions and PEP/PDP interoperability. Delegation chains, attenuation, revocation, and related capabilities remain the responsibility of capability systems or delegation model provided elsewhere in the architecture. As explained in the Overview chapter, an agent's required permissions cannot be fully known in advance, making least privilege dynamic. AuthZEN's access evaluation and search model is directly relevant to that problem because decisions can be made against the concrete action, resource, and context at execution time. Therefore, rather than representing a limitation, this can be considered a distinctive strength of AuthZEN, enabling it to be used without violating the requirements of agentic systems.

Scorecard

#	Requirement	Verdict	Notes
1	Accountable (agent vs. principal/operator)	Partial	AuthZEN core can identify a subject but does not standardize relationship between the responsible-principal and the agent. COAZ-MCP explicitly separates human subject and agent context, which improves agent accountability. However, it does not establish a delegation proof or legal-operator binding.

#	Requirement	Verdict	Notes
2	Resistant to confused deputy	Partial (deployment-dependent)	AuthZEN evaluates explicit Subject-Action-Resource tuples and supports resource-specific authorization decisions. COAZ and COAZ-MCP further bind authorization to protocol inputs and trust-anchored identities. However, the ecosystem lacks delegation artifacts and end-to-end cryptographic delegation chains, leaving confused-deputy resistance dependent on correct PEP behavior and deployment architecture.
3	Represent authorization policies (embed/link; "undelegatable", re-delegation rules, advisory "please")	No for core (out of scope); Partial with ecosystem	Policy language, architecture, and state management of the PDP are outside the scope of AuthZEN. The ecosystem improves policy-artifact portability and request-binding interoperability through Policy Store, COAZ, and COAZ-MCP. However, authorization-policy

#	Requirement	Verdict	Notes
			semantics remain policy-engine-specific and are not inherently interoperable across policy engines or organization.
4	Chainable	No	Verdict is "No" because it is not a delegation framework. Not because it is incompatible with delegation frameworks. Neither the core data model nor COAZ-MCP contains a normative parent grant, delegator, delegatee, delegation edge, parent capability, or proof-chain primitive. COAZ-MCP improves subject/agent attribution, but not multi-hop delegation. Access Evaluations support request batching (boxcarring) but not chaining. Each evaluation is processed independently and cannot depend on the outcome of another evaluation in the same request.
5	Cross-organizational / locally verifiable	Partial (deployment-dependent)	Enable cross-vendor permission request/decision

#	Requirement	Verdict	Notes
			interoperability. Cross-organizational use is possible through a trusted remote PDP, but AuthZEN does not standardize policy exchange or provide locally verifiable proof of delegated authority. The receiving organization relies on the remote PDP's decision.
6	Attenuated	No	The core API has no native delegation or attenuation model. The broader ecosystem also does not establish that downstream authority is a non-widening subset of upstream authority. Any attenuation must therefore be implemented by the underlying policy engine and data model, not by AuthZEN itself.
7	Self-revocable (holder or anyone in chain)	No	No native self-revocation of delegated authority. Revocation is a responsibility of surrounding policy/token/delegation systems. Policy

#	Requirement	Verdict	Notes
			changes can revoke access, but this is different from delegation-chain self-revocation by the delegator or holder.
+	Authentication / Proof of Possession	No (out of scope)	AuthZEN relies on external authentication mechanisms and does not define holder proof-of-possession for delegated authority.
+	Privacy of delegation chain	No (undefined)	There is no standardized delegation chain to disclose or hide. An implementation could store one entirely PDP-side and thereby hide it from the agent, but that is an architectural consequence, not an AuthZEN feature.
+	Offline-capable	No	AuthZEN is transport-agnostic and a PDP can be deployed locally. However, "Offline" in this context means the ability to create, carry, present, and verify delegated authority without contacting a PDP. Because AuthZEN

#	Requirement	Verdict	Notes
			defines neither a native delegation artifact nor an offline-verifiable delegation primitive, its Offline-capable verdict should be No.

Detailed Evaluation

1. Accountable (agent vs. principal/operator)

The Core can identify a Subject, defined as a "user or machine principal," through mandatory type and id fields and optional properties. An Access Evaluation API contains one and only one Subject. While that Subject may represent either a user or a machine principal, the specification provides no standard way to model additional Subjects, distinguish an acting agent from an accountable principal/operator, or express a standardized relationship between them. Context could carry operator, delegator, or chain information, but their semantics would be implementation-specific rather than interoperable. Therefore, an auditor cannot reliably reconstruct the accountable principal or hop-by-hop delegation solely from a conformant AuthZEN request. The specification also excludes policy architecture, state management, and API authentication from its scope.

The result is more favorable when considering the broader ecosystem rather than the Core specification alone.

With COAZ-MCP, the binding explicitly separates the human subject from the acting agent by allowing the human identity to populate subject.id while representing the agent in context.agent. As a result, auditors can determine not only that an agent acted, but also on whose behalf the authorization decision was evaluated. However, the specification does not define delegation relationships or delegation chains.

COAZ strengthens the trustworthiness of this model through mapping controls and trust-anchored fields. Bindings may require critical fields, such as subject identity, to be derived from independently verifiable inputs rather than caller-supplied data. This improves identity provenance and reduces the risk of agent-supplied identity assertions. Nevertheless, COAZ remains an identity-to-authorization mapping framework and does not define delegation relationships or delegation chains.

Thus, the ecosystem improves agent-versus-principal attribution and identity provenance, but it still falls short of providing interoperable delegation provenance and multi-hop accountability.

2. Resistant to confused deputy

AuthZEN's data model provides a structural foundation for confused-deputy resistance because every Access Evaluation request identifies a required Subject, Action, and Resource, with optional Context. Resource represents the target, while Action represents the intended operation and may contain operation parameters. Consequently, AuthZEN can ask "may this subject perform this specific action on this specific resource?" rather than merely checking whether an identity has a generic role. However, the API does not define how the PEP derives these values from the actual invocation. Subject, Resource, and Action attributes are supplied by the PEP, API authentication is out of scope, and enforcement remains the PEP's responsibility. Therefore, AuthZEN alone neither proves that the Resource/Action matches what the upstream requester designated nor ensures that the authority used for a downstream operation is the authority intended by the requester rather than the deputy's own ambient authority.

COAZ and COAZ-MCP provide procedural binding between an invocation, the constructed authorization request, and the resulting authorization decision, reducing the likelihood that designation is lost and authorization decisions become based on the deputy's ambient authority. COAZ requires bindings to specify the source and structure of input variables, treats mapping inputs as potentially untrusted, provides a mechanism for trust-anchored fields, and requires mapping inconsistencies or evaluation failures to be treated as mapping errors. COAZ-MCP binds MCP tool names, arguments, resource URIs, token claims, and server audience information into authorization requests. Its trust-anchored subject.id may be verified against the designated subject-identity claim, while declared mappings allow authorization policies to incorporate tool parameters directly.

Taken together, these ecosystem specifications strengthen the preservation of requester intent and reduce the risk that access decisions become detached from the actual operation being performed. However, they still lack a cryptographically transferable designation-and-authority artifact and end-to-end attenuating delegation chain. Consequently, confused-deputy resistance remains partial and deployment-dependent.

3. Represent authorization policies

Core specification does not standardize an authorization-policy representation. The Access Evaluation model provides SARC, but it does not define a standardized mechanism for embedding or referencing authorization policies themselves. Furthermore, the PDP's policy language, architecture, and state management are explicitly outside the scope of the specification. The specification illustrates advice, obligations, reasons, and step-up instructions as possible uses of Decision Context, but explicitly leaves their semantics and format outside the standard. Consequently, AuthZEN does not provide interoperable semantics for advisory rules, re-delegation constraints, or disclosure-governance policies.

Policy Store standardizes policy packaging and interchange, not policy semantics. Policies remain policy-engine-specific. Consequently, policy artifacts can be exchanged and discovered

through a common packaging mechanism, but the semantics of delegation, re-delegation rules, undelegatable authority, and other authorization policies are not inherently interoperable across policy engines or organizations, since their meaning and enforcement are still defined by the underlying policy engine and its policy model.

COAZ and COAZ-MCP standardize how operations are mapped to AuthZEN authorization requests (COAZ Mapping, COAZ-MCP Declaring a Mapping), not the semantics of authorization policies. COAZ provides declarative mappings from protocol inputs into AuthZEN SARC requests, while COAZ-MCP allows such mappings to be embedded in MCP tool definitions and enforced by a PEP before execution. Consequently, operations can be bound to authorization requests in a portable and interoperable manner, but the meaning of the policies being evaluated remains defined by the underlying PDP and policy engine rather than by COAZ or AuthZEN.

None of these specifications defines a common authorization-policy language or semantic model. They improve interoperability at the authorization interface layer (requests, responses, and policy artifacts), but not at the policy semantic layer. Consequently, different PDPs can exchange authorization requests and decisions through a common protocol, and policy artifacts may be shared through common packaging mechanisms, but the meaning and enforcement semantics of those policies remain implementation specific.

4. Chainable

If being chainable is interpreted as preserving delegated authority across multiple hops, AuthZEN and its current ecosystem do not provide a standardized delegation-chain model. Delegated authority remains external to the AuthZEN interoperability model. The same observation largely applies to provenance reconstruction. However, AuthZEN itself should not be interpreted as an impersonation-oriented model. Although it does not solve multi-hop delegation, neither does it force workflows into impersonation. Instead, AuthZEN can consume and evaluate externally provided delegation-related context as part of a delegated-authority workflow.

The Access Evaluations API should likewise not be interpreted as delegation chaining. It supports batching (boxcarring) of authorization requests, but each evaluation remains an independent authorization decision. Multiple evaluations do not create, transfer, attenuate, or propagate authority between actors. Consequently, request batching is fundamentally different from a genuine delegation chain and should not receive chainability credit.

The ecosystem specifications improve interoperability but do not materially change this conclusion. Policy Store, COAZ, and COAZ-MCP improve portability, interoperability, and attribution, but none of them define delegated authority as a first-class, verifiable artifact across multiple hops. Nevertheless, like AuthZEN itself, they do not inherently force delegated-authority workflows into impersonation.

AuthZEN Issue #416 "[Feature Request] Add 'delegation_chain' to Subject for AI Agent and Multi-Actor Scenarios" further illustrates this point, which proposes introducing a delegation_chain structure for AI-agent scenarios. The proposal explicitly argues that the current subject model cannot adequately represent multi-hop delegation chains involving users, agents, and downstream services. This should not be interpreted as existing support. Instead, it demonstrates that delegation provenance and multi-hop authority flow are recognized gaps in the current model.

Accordingly, if chainability is viewed as a property of delegated authority itself rather than authorization interoperability, AuthZEN should still receive No. Nevertheless, the reason is not that AuthZEN prevents delegation chains or requires impersonation. AuthZEN deliberately operates at a different architectural layer. It can evaluate and enforce authorization decisions informed by delegated-authority context, but the delegation chain itself, including attenuation, provenance, chain verification, and revocation semantics, must be supplied by another system. AuthZEN is therefore best characterized as a consumer of delegation chains rather than a provider of delegation chains.

5. Cross-organizational / locally verifiable

The Core enables authorization interoperability across organizational boundaries and the ecosystem improves it, but participating organizations must separately establish trust in the relevant token issuers, identity claims, and PDPs. (COAZ Trust-Anchored fields improve input trustworthiness but do not establish cross-organizational trust.) The Core does not define how such cross-organizational trust is established, and policy semantics remain external to the AuthZEN ecosystem.

The Access Evaluation API returns an access decision, but not the underlying proof needed for another organization to independently validate why that decision was made. Nor do Policy Store, COAZ, or COAZ-MCP define such an authorization-proof format. AuthZEN ultimately relies on an access decision made by a PDP.

Consequently, an organization can operate its own PDP locally, and organizations can integrate their authorization systems across organizational boundaries, but neither outcome constitutes local verification of authorization evidence. The receiving organization still relies on the PDP that made the access decision and on the separately configured identity and token trust infrastructure, rather than independently verifying a transferable delegation or authorization artifact. The criterion is therefore rated Partial.

Cross-organizational deployments can be strengthened by combining AuthZEN with external trust frameworks for issuer trust, identity federation, key discovery, delegation credentials, and shared policy semantics, while continuing to use AuthZEN as the interoperable authorization layer connecting otherwise independent trust and policy domains.

6. Attenuated

The Core alone can express narrowly scoped, context-sensitive access questions through SARC, making it well suited to runtime least-privilege enforcement. However, as with the previous sections, given the scope of the specification, neither monotonic attenuation nor the prevention of authority widening across delegation hops is defined by AuthZEN itself.

Adding the three ecosystem specifications does not improve support for attenuation. As discussed in the previous sections, the Policy Store does not establish parent-child delegation semantics or a monotonic subset invariant. COAZ strengthens the trustworthy construction of fine-grained authorization inputs, but it explicitly preserves AuthZEN request semantics rather than defining delegation semantics. COAZ-MCP extends this mapping framework to MCP, and its controls can enforce constraints supplied by policy. However, none of these specifications establishes that a child delegation cannot exceed the authority of its parent.

Thus, AuthZEN provides strong runtime enforcement for attenuated authority defined and validated elsewhere, but it does not provide native delegation attenuation.

7. Self-revocable

The core specification provides no protocol mechanism by which a holder or an ancestor in a delegation chain can revoke a specific delegated authority, nor does it define any requirement for revocation to cascade to descendants. While changes to PDP policy may cause subsequent evaluations to return false, the specification does not define who is authorized to make such changes or whether that authority derives from participation in a delegation chain.

The specification also defines no revocation-state object, revocation-propagation protocol, cache-invalidation mechanism, network-partition behavior, or maximum revocation latency. Although expiration timestamps or other short-lived authorization evidence may be supplied as contextual input by an implementation, this represents expiration rather than explicit revocation and is not standardized by the API.

The ecosystem does not add delegation-chain self-revocation capabilities. The Policy Store specification allows administrators to upload immutable, versioned Policy Store packages through a POST-only Policy Store API, while existing policy stores cannot be updated or deleted. This supports policy deployment, versioning, and rollback, but not holder- or ancestor-initiated revocation. COAZ defines how protocol-specific information models are mapped into AuthZEN authorization requests, but it introduces neither delegation lineage nor revocation operations. Similarly, COAZ-MCP does not define holder- or ancestor-driven revocation, child-selective revocation, cascading revocation semantics, or guarantees regarding cache consistency, revocation propagation, or network partitions.

8. Authentication / Proof of Possession

Authentication of the Authorization API is explicitly out of scope. While OAuth 2.0 support is RECOMMENDED, OAuth 2.0 deployments are typically based on bearer tokens and therefore do not inherently provide holder proof-of-possession. Achieving proof-of-possession generally requires additional mechanisms such as DPoP, mTLS, sender-constrained tokens, or comparable cryptographic binding techniques. AuthZEN neither mandates nor standardizes any of these mechanisms. Consequently, while AuthZEN can operate in deployments that provide proof-of-possession through external identity and transport layers, the specification itself does not define holder-bound credentials or cryptographic proof-of-possession for delegated authority.

The ecosystem improves the integrity and trustworthiness of identity-related inputs, but it still does not satisfy the proof-of-possession criterion. COAZ can designate trust-anchored fields that must be derived from trusted inputs or verified by the PEP. COAZ-MCP uses decoded JWT OAuth access-token claims and recommends anchoring subject.id to the designated subject-identity claim. It also keeps agent identity separately in context.agent. These mechanisms reduce the risk of identity-claim substitution, but they do not require sender-constrained tokens, a signed invocation, or proof that the agent possesses a private key bound to delegated authority. The Policy Store packages trusted-issuer configuration and policy artifacts, but it likewise does not establish holder proof-of-possession for delegated authority.

9. Privacy of delegation chain

AuthZEN API 1.0 does not satisfy this criterion because the specification does not define a delegation-chain representation and therefore provides no standardized mechanism for concealing, selectively disclosing, or verifying delegation chains. While an implementation could include delegation-related information in custom properties or context fields, the semantics and privacy characteristics of such information remain implementation-specific rather than standardized by AuthZEN.

Including the ecosystem specifications does not change this conclusion. Neither COAZ nor COAZ-MCP defines a multi-hop delegation chain or any mechanism for hiding selected hops from the MCP client or intermediary. Any delegation-chain privacy that may be achieved through a particular deployment architecture is implementation-specific rather than standardized by the AuthZEN ecosystem.

Accordingly, AuthZEN does not satisfy this criterion. More precisely, the property is undefined because no delegation-chain representation exists within the specification.

10. Offline-capable

The Core does not satisfy this criterion because the specification does not define a self-contained delegated-authority artifact that can be created, delegated, presented, and independently verified without PDP involvement. The normal AuthZEN model is the evaluation

of an authorization request by a PDP at runtime. Although the specification is transport-agnostic and permits locally deployed or embedded PDPs, local communication is not equivalent to offline delegation. The ability to evaluate authorization decisions without an external network connection should not be confused with the ability to create, transfer, and verify delegated authority offline.

The same conclusion applies when the ecosystem specifications are considered. Neither Policy Store, COAZ, nor COAZ-MCP defines a self-contained delegated-authority artifact that can be presented and independently verified outside the PDP evaluation model. Consequently, the ecosystem provides no standardized basis for offline delegation. While some deployments may reduce network dependencies through locally available policy artifacts or locally deployed PDPs, runtime authorization remains dependent on a PDP or policy engine rather than a self-verifying delegation artifact.

More precisely, locally deployed or disconnected PDPs may support offline policy evaluation, but they do not provide offline delegation.

AuthZEN-specific Considerations

1. Treat Policy Store portability carefully

AuthZEN standardizes the container and deployment unit for policies, schemas, entities, and issuer configuration, enabling these artifacts to be packaged, exchanged, and reused across implementations. However, Policy Store portability should not be confused with delegation interoperability. While AuthZEN standardizes the packaging and exchange of policy artifacts, the semantics of delegation, attenuation, and revocation remain specific to the underlying capability system implementation and architecture. Consequently, a policy store may be portable between systems, while the resulting delegation behavior may differ across authorization engines and deployments.

2. Search results should not be interpreted as proof of capability or delegation

AuthZEN Search is designed for discovery rather than for conveying authority. The specification states that "Search APIs provide lists of resources, subjects or actions that would be allowed access." Search results indicate what may be permitted under the PDP's current state, but they do not by themselves constitute proof of delegated authority, capability possession, or authorization grants.

3. Search can create an information-disclosure risk

Reverse-query APIs inherently expose information derived from the current state of the authorization system. If made available to insufficiently trusted callers, repeated or carefully crafted searches may reveal the existence of resources, access relationships, or characteristics

of effective authorization policies. While Search can improve usability and discovery, it can also increase the risk of information disclosure. Appropriate authentication, query scoping, rate limiting, response minimization, and careful handling of diagnostic information should therefore be considered.

4. Policy Store administration

Policy Store specification does not define a normative authorization model for administration of the Policy Store itself. Section 13, "Security Considerations," indicates that security, authorization, and operational controls of Policy Store itself are implementation responsibilities rather than standardized Policy Store behavior. Specifically, the specification does not prescribe who may create, modify, approve, publish, or delete policy bundles, nor does it define administrative roles, delegation models, or approval workflows for policy governance. These responsibilities are left to the implementing environment and its operational controls. As a result, organizations may adopt different approaches, ranging from repository-based access controls and CI/CD approval processes to dedicated PAP solutions. This separation preserves implementation flexibility, but it also means that governance, segregation of duties, and administrative access control for the Policy Store remain implementation-specific rather than standardized by AuthZEN.

5. Decision Interoperability vs. Policy Interoperability

The Authorization API standardizes how a PEP requests and receives access decisions from a PDP, but it does not standardize policy languages, policy semantics, or policy execution models. A conformant PDP may use Cedar, Rego, Zanzibar-style relationships, or other policy systems, while exposing the same AuthZEN interface. AuthZEN achieves interoperability through interoperable authorization requests and access decisions, rather than through interoperability of policy semantics. One organization can consume access decisions produced by another organization's PDP, but AuthZEN does not require PDPs to share, exchange, interpret, or enforce one another's policies.

Summary

AuthZEN can serve as a strong runtime authorization and policy-decision layer within an agentic authorization stack. Its primary value lies in standardizing authorization decisions and interoperability between authorization components, while supporting emerging AI-agent and tool-invocation scenarios through initiatives such as COAZ and COAZ-MCP.

However, AuthZEN is not a delegated-authority system. Delegation semantics, including delegation proof, attenuation, and revocation, remain outside the scope of Authorization API 1.0 and are provided by other components of the overall architecture. Likewise, policy representation is intentionally left to the underlying PDP implementation.

Understanding these boundaries is important when comparing AuthZEN with policy frameworks such as Cedar and delegation-focused approaches such as UCAN and zCAP.

References

- OpenID AuthZEN Authorization API 1.0 Specification
- OpenID AuthZEN Policy Store Format Specification
- COAZ: A Framework for Mapping Information Models to AuthZEN Authorization Requests - Draft 1
- COAZ-MCP: COAZ Binding for the Model Context Protocol - Draft 1
- AuthZEN GitHub Issue #416 "[Feature Request] Add 'delegation_chain' to Subject for AI Agent and Multi-Actor Scenarios"
- OpenID AuthZEN Meeting Notes
- AuthZEN Interop

Synthesis -- Comparing the Surveyed Specifications

The preceding chapters evaluated each specification on its own terms. This chapter puts them side by side: first a taxonomy that explains why the specs are so different in shape, then the merged scorecard, then the two comparisons the scorecard raises (server-mediated versus certificate delegation, and the policy decision family as a complement rather than a competitor), and finally the gaps that no surveyed spec fills. Those gaps are the working group's candidate agenda.

Three Families

The surveyed specs are not six competitors for one job. They fall into three families, distinguished by where authority lives:

- Certificate capabilities (zCaps, UCANs). Authority lives in a portable, signed artifact. The delegatee holds it, can attenuate and re-delegate it offline, and presents it with proof of key possession. The verifier checks the chain locally; no authorization server exists.
- Server-mediated authorization (the OAuth 2.0/2.1 lineage, AAuth). Authority lives in a server: the OAuth Authorization Server, or AAuth's Person Server with its mission state. Tokens reference that authority rather than embodying it, and delegation, attenuation, and revocation are acts performed at or by the server.
- Policy decision systems (Cedar, AuthZEN). Authority lives in the verifier's policy store. There is no token of authority at all; the request presents identity evidence and the verifier evaluates rules it already holds. Cedar is the family's policy language. AuthZEN is its wire protocol, standardizing how an enforcement point (PEP) asks a decision point (PDP) for a verdict. The two cover complementary layers of the same architecture, much as zCaps and UCANs do for the certificate family.

The families also differ in which of the specification types from the Introduction (data model, protocol, possession mechanism, chainable proof methods) each spec actually covers. Much of

the apparent unevenness between chapters reflects this: the chapters describe different layers of a potential stack, not six answers to the same question.

Spec	Data model	Protocol	Possession mechanism	Chainable proofs	Policy language
OAuth 2.0 / 2.1	Partial (scopes, JWT claims)	Yes (core focus)	Optional (DPoP, mTLS)	No (Token Exchange emulates, online)	No (opaque scope strings)
AAuth	Partial (act claim, tokens)	Yes (core focus)	Yes (signed requests, core)	Server-mediated (Mission Log)	No (deferred to PS policy)
zCaps	Yes (core focus)	Out of scope (invocation and revocation conventions only)	Yes (HTTP signatures, Data Integrity)	Yes (core focus)	Extension point (caveat, reserved)
UCANs	Yes (core focus)	Partial (invocation and revocation sub-specs)	Yes (signed envelope)	Yes (core focus)	Extension point (pol syntax, no vocabulary)
Cedar	No (no authorization artifact)	No	No	No	Yes (core focus)
AuthZEN	Partial (SARC request shape; no authorization artifact)	Yes (core focus: PEP/PDP evaluation and search)	No (out of scope)	No	No (explicitly out of scope; engine-specific)

Data Model Correspondence

The family taxonomy is easiest to see at the level of concrete fields. The table below maps each spec's terminology onto the common data model from the Overview: despite the differences in encoding, the same shape recurs wherever a spec defines an authorization artifact at all. (Cedar is absent because it defines no such artifact; its equivalents are schema entities in the

verifier's store. AuthZEN's column reflects the request shape a PEP submits, not an authority artifact, per its chapter. GNAP is included for field comparison, although it is not otherwise surveyed in this paper.)

Common field	**OAuth 2.1**	**AuthZEN**	**GNAP**	**zCaps**	**UCANs**	**AAuth**
Agent	client_id	context.agent (COAZ-MCP); no agent concept in core	client (rich object)	controller (DID)	Delegation: aud, Invocation: iss	agent identifier (agent_token subject, stable across key rotation)
Principal	sub	subject	subject	implicit (in delegation chain)	Delegation: iss, Invocation: sub	person, vouched by the Person Server
Invocation / key binding	Optional, via DPoP	no	Yes (DPoP, HTTPSig, mTLS)	Yes: HTTP signatures plus Capability-Invocation header, or Data Integrity proof	Yes: its own signed-envelope mechanism; keys resolved from DIDs	Yes: HTTP Message Signatures on every request; cnf binds tokens to the current key
Resource	scope or RAR (structured scope); see also htm/htu claims	resource and context	locations and datatypes	invocation Target (URL or object)	specified in args	resource_token (names resource and scope; optional account parameter)
Actions	scope (also htm)	actions	actions	allowedAction	cmd	scope (in resource and auth tokens)

Common field	OAuth 2.1	AuthZEN	GNAP	zCaps	UCANs	AAuth
Caveats / additional policy	n/a	context	?	caveat (reserved, unused in deployment)	pol (policy statements) plus args	none in the token; mission context evaluated server-side at the PS
Delegation chain	only via Token Exchange	no	no	Yes: capabilityChain proof chains	Yes: proof chains (CID-linked)	act claim plus the server-held Mission Log
Protocol (how is authorization requested?)	OAuth 2.1 itself	AuthZEN	GNAP	out of scope (uses GNAP, VCALM, and similar)	out of scope (transport left open)	AAuth itself
Revocation	RFC 7009	n/a (Policy Store rollback is versioning, not revocation)	n/a	Yes: out of band, at the RS	Yes: Revocation sub-spec; also a revocation capability	specified scenarios, exercised by the issuing servers; tokens expire within the hour

Two readings of this table reinforce the scorecard that follows. Empty and "n/a" cells are findings, not omissions: OAuth simply has no place to put a caveat or a chain, which predicts its rows 3 and 4 below. And the two cells where AAuth diverges from the certificate pattern (chain and caveats both live at the server, not in the token) are precisely what produces its server-mediated column throughout.

The Merged Scorecard

Verdicts below are taken from the per-spec chapters, which carry the detailed reasoning; the notes here are compressed to one line. zCaps and UCANs are shown separately for completeness, but as their chapter establishes, they score as a family. Cedar and AuthZEN likewise share a family, though they cover different layers of it and their verdicts diverge accordingly.

#	Requirement	OAuth 2.0/2.1	AAuth	zCaps	UCANs	Cedar	AuthZEN
1	Accountable (agent vs. principal/ operator)	Yes with asymmetric client auth; weaker with shared client_secret	Yes: distinct authenticated roles, act claim	Partial: signed edges, but principals are opaque DIDs	Partial: same as zCaps	Yes at the modeling level; not a cryptographic delegation edge	Partial: COAZ-MCP separates subject.id from context.agent; no delegation proof or operator binding
2	Resistant to confused deputy	No: scopes attenuate but do not designate	Partial: token modes bind designation; identity-based mode is ambient	Yes, by construction	Yes, by construction	No: identity-based, mitigation only	Partial: SARC carries designation per request; no delegation artifact, PEP-dependent
3	Represent authorization policies	Partial: scopes and RAR; extension-dependent	Partial: policy intentionally outside the protocol	Partial: caveat slot reserved, unused, no	Partial: pol syntax defined, no vocabulary	Yes, except the advisory tier; core competence	Partial: Policy Store packages artifacts; semantics stay

#	Requirement	OAuth 2.0/2.1	AAuth	zCaps	UCANs	Cedar	AuthZEN
				vocabulary			engine-specific (core: out of scope)
4	Chainable	No: single-hop; Token Exchange emulates multi-hop online	Yes, server-mediated: act claim plus Mission Log; sub-agent nesting single-level	Yes: parentCapability plus capabilityChain	Yes: hash-linked proof chains	No: no chain artifact; stack-dependent	No: consumer of delegation chains, not a provider
5	Cross-organizational / locally verifiable	Conditional: JWT access tokens yes, opaque tokens no	Yes: cross-domain by design; trust via configured relationships	Yes: local cryptographic verification, DID principals	Yes: same, via CID resolution	Partial: evaluation local, policy store unilateral	Partial: cross-vendor decisions interoperate; relying party trusts the remote PDP
6	Attenuated	Via the AS only (scope narrowing, Token Exchange); not by the holder	Partial, differs by mode: sub-agent yes, call chaining intentionally no	Yes: monotonic, verifier-enforced	Yes: monotonic, verifier-enforced	Partial: forbid-composition shrinks authority, but no subset check between grants	No: attenuation left to the underlying policy engine

#	Requirement	OAuth 2.0/2.1	AAuth	zCaps	UCANs	Cedar	AuthZEN
7	Self-revocable (holder or chain)	Partial: RFC 7009 plus short lifetimes; no chain concept	Partial: revocation by the servers holding authority, not the holder	Yes, at the RS revocation endpoint	Yes, via the Revocation sub-spec	Partial: policy store change by the administrator	No: policy change is administrative, not holder revocation
+	Authentication / Proof of Possession	Optional (DPoP, mTLS); bearer remains the deployed default	Yes: every request signed, core mechanism	Yes: HTTP signatures or Data Integrity proofs	Yes: signed envelope, nonce, time bounds	No: out of scope; evidence-issuance checks only	No: out of scope; bearer OAuth is the recommended default
+	Privacy of delegation chain	High by absence: no chain exists to disclose	Partial: history held at the PS, disclosure deployment-defined	No: full chain visible to carrier and verifier	No: same	Possible by architecture: carrier sees nothing, verifier sees everything	No (undefined): no chain representation exists
+	Offline-capable	Partial: issuance online; JWT verification offline	Partial: verification survives on cached keys; obtaining authority online	Yes for delegation and verification	Yes for delegation and verification	Partial: evaluation offline; granting online and administrative	No: local PDP evaluation is not offline delegation

Reading the matrix

Four patterns are worth drawing out, because they organize everything else in this chapter.

The capability family owns the chain rows. On the criteria that concern the mechanics of delegated authority (confused deputy, chainable, cross-organizational, attenuated, self-revocable: rows 2 and 4 through 7), zCaps and UCANs are the only column pair that is Yes across the board, and they achieve it offline and holder-side. Every other column pays for those rows with a server dependency, a partial verdict, or both. This is the clearest architectural line in the survey, and it is a family property, not a property of either spec individually.

The identity side owns the semantic rows. On accountability (row 1) and policy expression (row 3), the verdicts invert. The systems that know who everyone is (AAuth's authenticated roles, Cedar's typed principals, OAuth clients with asymmetric auth) can distinguish agent from operator natively; the capability family deliberately keeps principals opaque and delegates that distinction to a companion identity layer. AuthZEN's COAZ-MCP binding follows the identity side's pattern, separating the human subject from the acting agent, though without a delegation proof it stops at attribution. Likewise the richest policy expression in the survey (Cedar) sits in the one spec with no delegation mechanics at all, while the specs with delegation mechanics reserve policy extension points and leave them empty.

Chain privacy inverts too, and for a structural reason. The capability family's defining artifact, the portable chain, is exactly what its privacy row suffers from: the carrier and the verifier both see the whole provenance log. The systems with no portable chain get carrier-side privacy for free, at the cost of an all-seeing server or policy store. Neither side offers the checklist's actual wish (principal and verifier know all, intermediate carrier does not) as a mechanism; the closest approximations are workarounds (ephemeral keys) or deployment choices (PS disclosure policy).

No column is complete. Every spec has at least one No or Partial in the required rows 1 through 7. The capability family misses semantics (rows 1 and 3); the server-mediated family misses holder-side and offline mechanics (rows 4, 6, 7 in various ways); the policy decision family misses the delegation artifact entirely (row 4, and for Cedar row 2 as well; AuthZEN's SARC request carries designation with each decision, which earns it a Partial on row 2 that pure identity evaluation does not get). The AuthZEN column is the sparsest in the matrix, and its chapter is explicit about why: the spec addresses the decision interface layer, so its No verdicts mark a different layer rather than a failed attempt at delegation. This is the observation that motivates both the delegation-stack section and the gaps section below.

Server-Mediated vs. Certificate Delegation

AAuth and the certificate capability family aim at the same use case (autonomous agents acting across organizational boundaries under delegated, revocable, auditable authority) and give opposite answers to the central question of where authority lives. The contrast is the sharpest one available in this survey, precisely because so much else about them agrees: both reject

bearer tokens for signed requests, both identify agents by key material, both record who delegated what to whom.

- Portability. A certificate capability is self-contained: the chain travels with the invocation and any verifier can check it. AAuth's delegation history lives in the Person Server's Mission Log; verification of standing depends on authenticated interaction with the server that holds the state.
- Offline delegation. The capability family delegates by signing, with no round trip. AAuth has no offline delegation; obtaining and extending authority are online protocol exchanges. This is the efficiency and partition-tolerance line.
- Attenuation. The capability family enforces algebraic, monotonic attenuation at every link: a child provably holds a subset. AAuth deliberately rejects the subset rule for call chaining; the Person Server evaluates each hop against mission context, which its specification argues is more flexible than algebraic rules. This is a genuine philosophical split (constraint by algebra versus constraint by governance), not an omission on either side.
- Revocation locus. In the capability family, anyone in the chain can revoke, and the enforcement point learns of it. In AAuth, the servers holding authority revoke; the holder of a credential cannot act on the credential itself.
- Key lifecycle. AAuth delegations reference stable agent identifiers, with the cnf claim binding tokens to the current key; identity survives key rotation, at the cost of lifecycle questions about delegations that straddle a rotation. The capability family binds delegations to keys directly, and typically sidesteps rotation with short-lived, delegation-specific keys.

For agentic delegation, the trade reduces to this: the certificate model gives stronger guarantees (offline, locally verifiable, algebraically attenuated, holder-revocable) at the price of thin semantics, while the server-mediated model gives richer semantics and governance (missions, consent, authenticated roles) at the price of a live, trusted server on the path of authority.

The Policy Decision Family as Complement

Cedar is, almost row by row, the complement of the capability family: strongest exactly where zCaps and UCANs are weakest (policy expression, agent-vs-operator modeling, fine-grained revocation, composition of authority) and weakest exactly where they are strongest (no chain, no portable grant, no holder-side anything). Read as a competitor it fails the survey; read as a component it fills the survey's emptiest cell.

The specific fit: both zCaps and UCANs reserve a policy extension point (caveat; pol) and ship no vocabulary for it. Cedar is a mature, analyzable policy language with no carrier. A capability chain whose caveats were expressed in a Cedar-class language would give the family what its chapters identify as their highest-value missing piece, with one property nothing else in the survey offers: because Cedar is non-Turing-complete and formally analyzable, delegation policies could be audited and subset relationships proven offline, at delegation time rather than

invocation time. The reverse composition also holds: Cedar deployments that need portable, cross-organizational grants need exactly the carrier the capability family provides.

AuthZEN extends the same complement to the protocol layer, and makes the composition concrete. Cedar has no standard request/decision interface; AuthZEN supplies exactly that. In a composed stack, a capability verifier can act as a PEP: it checks the chain and proof of possession locally, then consults a PDP over AuthZEN for the caveat evaluation the capability family leaves unspecified. COAZ points the same direction from the other side, mapping live protocol operations (including MCP tool calls, via COAZ-MCP) into decision requests that reflect the actual invocation parameters. None of this requires the capability chain to change shape; it fills the empty caveat slot with an evaluation pipeline that already exists.

This suggests the shape of a composed delegation stack, assembled from surveyed parts:

1. Entry point (identity layer): an identity system (OIDC, Verifiable Credentials) makes the human or organizational judgement call and binds operator to agent; the zCap specification itself recommends this split.
2. Authority flow (carrier layer): a certificate capability chain carries the delegated, attenuated authority across hops and organizations, offline and locally verifiable.
3. Restrictions (policy layer): caveats within the chain expressed in a standardized, analyzable policy language, evaluated at each hop and provable between hops; where evaluation is handed to a PDP, AuthZEN is the standard interface for asking.
4. Presentation (possession layer): signed invocation of the transport in use (HTTP Message Signatures, Data Integrity proofs, DPoP-style binding), with keys held outside the agent per the Introduction's requirements.

No surveyed spec provides more than two adjacent layers of this stack, and no pair of them currently interoperates by standard. That observation leads directly to the gaps.

Gaps

Reading down the merged matrix, the recurring Partial and No cells cluster into six gaps that no surveyed specification fills. These are candidates for working group attention, ordered roughly by how directly they block agentic delegation.

1. A standard policy and caveat vocabulary for delegation control. Both certificate capability specs have the extension point and neither ships vocabulary; AAuth places policy outside the protocol; OAuth scopes are opaque strings; AuthZEN standardizes the decision request while leaving policy semantics to the engine. The concrete missing terms are the checklist's own examples: undelegatable, re-delegation constraints, delegation depth. Cedar demonstrates that such a vocabulary can be expressive and formally analyzable at once. This is the gap where existing pieces come closest to composing, and arguably the highest-value item.

2. An advisory tier. No surveyed system can express a request whose violation is not refused ("please do not re-delegate outside the org"): every evaluation model in the survey is binary enforce/reject. Advisory semantics matter for agentic delegation because agents cross governance boundaries where hard enforcement is impossible but recorded intent still has value (for audit, for liability, for downstream policy).
3. Agent-vs-operator semantics and identity binding. The capability family cannot say which principal is the accountable legal entity; the identity-centric systems can, but only inside their own trust domain. What is missing is a standardized companion mechanism (the VC-based binding both capability chapters gesture at) so that accountability to an operator survives cross-organizational, multi-hop delegation without collapsing back into identity-based authorization. AuthZEN's open Issue #416, which requests a delegation_chain structure in the Subject for AI-agent scenarios, is this same gap filed independently by an adjacent working group.
4. Delegation chain privacy. The checklist's formulation (principal and verifier know the full story, the intermediate carrier does not) is offered by no spec as a mechanism. The capability family discloses everything to everyone on the path; the server-mediated and policy families relocate the problem to an all-seeing server. Selective disclosure or blinding of chain interiors is an open design problem.
5. Offline and decentralized revocation. Every surveyed revocation mechanism is state at an enforcement point that must be reached: an RS endpoint, a server's grant, a policy store. For the local-machine and LAN agentic use cases the checklist highlights, revocation semantics under partition are unspecified everywhere. UCAN's open dissemination model (gossip, DHT) is the closest starting point, but its guarantees are undefined.
6. Cross-organizational policy exchange. Even where evaluation is local, policy semantics are unilateral: Cedar's store belongs to one verifier, an AS's scope definitions to one provider, a PS's mission policy to one server. There is no standard by which one organization hands another a policy fragment with agreed meaning. AuthZEN's Policy Store draft is a partial answer: it standardizes the packaging and exchange of policy artifacts while explicitly leaving their semantics engine-specific, sharpening this gap without closing it. This is the cross-organizational criterion's unmet half, and it compounds gap 1: a shared caveat vocabulary is also the natural interchange format.

Conclusion

The survey's answer to "which specification should agentic delegation use" is that the question cannot have a general answer. The certificate capability family supplies the only viable backbone for the authority flow itself as first-class data element and throughline: it alone meets the chain criteria, offline and holder-side, and it alone is immune to the confused deputy by construction. But it is a backbone, not a solution; its own chapters identify the missing semantics, and those semantics are exactly what the identity-centric and policy systems in this survey already know how to express. The near-term work is therefore not a seventh competing spec but composition: a standardized policy vocabulary that can ride in a capability chain, a standardized identity binding at the chain's root (whether this be centralized in a “Mission Log”

or not), a standard decision interface (which AuthZEN already supplies) where caveat evaluation is handed to a policy engine, and revocation and privacy mechanisms designed for the partitioned, multi-organizational environments agents actually operate in. The gaps list above is, in effect, the specification outline for that work.

input: Synthesis chapter

Synthesis: Comparing the Surveyed Specifications

The preceding chapters evaluated each specification on its own terms. This chapter puts them side by side: first a taxonomy that explains why the specs are so different in shape, then the merged scorecard, then the two comparisons the scorecard raises (server-mediated versus certificate delegation, and the policy engine as a complement rather than a competitor), and finally the gaps that no surveyed spec fills. Those gaps are the working group's candidate agenda.

Three Families

The surveyed specs are not five competitors for one job. They fall into three families, distinguished by where authority lives:

- Certificate capabilities (zCaps, UCANs). Authority lives in a portable, signed artifact. The delegatee holds it, can attenuate and re-delegate it offline, and presents it with proof of key possession. The verifier checks the chain locally; no authorization server exists.
- Server-mediated authorization (the OAuth 2.0/2.1 lineage, AAuth). Authority lives in a server: the OAuth Authorization Server, or AAuth's Person Server with its mission state. Tokens reference that authority rather than embodying it, and delegation, attenuation, and revocation are acts performed at or by the server.
- Policy engine (Cedar). Authority lives in the verifier's policy store. There is no token of authority at all; the request presents identity evidence and the verifier evaluates rules it already holds.

The families also differ in which of the specification types from the Introduction (data model, protocol, possession mechanism, chainable proof methods) each spec actually covers. Much of the apparent unevenness between chapters reflects this: the chapters describe different layers of a potential stack, not five answers to the same question.

Spec	Data model	Protocol	Possession mechanism	Chainable proofs	Policy language
OAuth 2.0 / 2.1	Partial (scopes, JWT claims)	Yes (core focus)	Optional (DPoP, mTLS)	No (Token Exchange emulates, online)	No (opaque scope strings)

AAuth	Partial (act claim, tokens)	Yes (core focus)	Yes (signed requests, core)	Server-mediated (Mission Log)	No (deferred to PS policy)
zCaps	Yes (core focus)	Out of scope (invocation and revocation conventions only)	Yes (HTTP signatures, Data Integrity)	Yes (core focus)	Extension point (caveat, reserved)
UCANs	Yes (core focus)	Partial (invocation and revocation sub-specs)	Yes (signed envelope)	Yes (core focus)	Extension point (pol syntax, no vocabulary)
Cedar	No (no authorization artifact)	No	No	No	Yes (core focus)

Data Model Correspondence

The family taxonomy is easiest to see at the level of concrete fields. The table below maps each spec's terminology onto the common data model from the Overview: despite the differences in encoding, the same shape recurs wherever a spec defines an authorization artifact at all. (Cedar is absent because it defines no such artifact; its equivalents are schema entities in the verifier's store. AuthZEN and GNAP are included for field comparison, although they are not otherwise surveyed in this paper.)

Common field	**OAuth 2.1**	**AuthZEN**	**GNAP**	**zCaps**	**UCANs**	**AAuth**

Agent	client_id	n/a (no agent concept)	client (rich object)	controller (DID)	Delegation: aud, Invocation: iss	agent identifier (agent_token subject, stable across key rotation)
Principal	sub	subject	subject	implicit (in delegation chain)	Delegation: iss, Invocation: sub	person, vouched by the Person Server
Invocation / key binding	Optional, via DPoP	no	Yes (DPoP, HTTPSig, mTLS)	Yes: HTTP signatures plus Capability-Invocation header, or Data Integrity proof	Yes: its own signed-envelope mechanism; keys resolved from DIDs	Yes: HTTP Message Signatures on every request; cnf binds tokens to the current key
Resource	scope or RAR (structured scope); see also htm/htu claims	resource and context	locations and datatypes	invocationTarget (URL or object)	specified in args	resource_token (names resource and scope; optional account parameter)
Actions	scope (also htm)	actions	actions	allowedAction	cmd	scope (in resource and auth tokens)

Caveats / additional policy	n/a	context	?	caveat (reserved, unused in deployment)	pol (policy statements) plus args	none in the token; mission context evaluated server-side at the PS
Delegation chain	only via Token Exchange	no	no	Yes: capabilityChain proof chains	Yes: proof chains (CID-linked)	act claim plus the server-held Mission Log
Protocol (how is authorization requested?)	OAuth 2.1 itself	AuthZEN	GNAP	out of scope (uses GNAP, VCALM, and similar)	out of scope (transport left open)	AAuth itself
Revocation	RFC 7009	n/a	n/a	Yes: out of band, at the RS	Yes: Revocation sub-spec; also a revocation capability	specified scenarios, exercised by the issuing servers; tokens expire within the hour

Two readings of this table reinforce the scorecard that follows. Empty and "n/a" cells are findings, not omissions: OAuth simply has no place to put a caveat or a chain, which predicts its rows 3 and 4 below. And the two cells where AAuth diverges from the certificate pattern (chain

and caveats both live at the server, not in the token) are precisely what produces its server-mediated column throughout.

The Merged Scorecard

Verdicts below are taken from the per-spec chapters, which carry the detailed reasoning; the notes here are compressed to one line. zCaps and UCANs are shown separately for completeness, but as their chapter establishes, they score as a family.

#	Requirement	OAuth 2.0/2.1	AAuth	zCaps	UCANs	Cedar
1	Accountable (agent vs. principal/operator)	Yes with asymmetric client auth; weaker with shared client_secret	Yes: distinct authenticated roles, act claim	Partial: signed edges, but principals are opaque DIDs	Partial: same as zCaps	Yes at the modeling level; not a cryptographic delegation edge
2	Resistant to confused deputy	No: scopes attenuate but do not designate	Partial: token modes bind designation; identity-based mode is ambient	Yes, by construction	Yes, by construction	No: identity-based, mitigation only
3	Represent authorization policies	Partial: scopes and RAR; extension-dependent	Partial: policy intentionally outside the protocol	Partial: caveat slot reserved, unused, no vocabulary	Partial: pol syntax defined, no vocabulary	Yes, except the advisory tier; core competence

4	Chainable	No: single-hop; Token Exchange emulates multi-hop online	Yes, server-mediated: act claim plus Mission Log; sub-agent nesting single-level	Yes: parentCapability plus capabilityChain	Yes: hash-linked proof chains	No: no chain artifact; stack-dependent
5	Cross-organizational / locally verifiable	Conditional: JWT access tokens yes, opaque tokens no	Yes: cross-domain by design; trust via configured relationships	Yes: local cryptographic verification, DID principals	Yes: same, via CID resolution	Partial: evaluation local, policy store unilateral
6	Attenuated	Via the AS only (scope narrowing, Token Exchange); not by the holder	Partial, differs by mode: sub-agent yes, call chaining intentionally no	Yes: monotonic, verifier-enforced	Yes: monotonic, verifier-enforced	Partial: forbid-composition shrinks authority, but no subset check between grants
7	Self-revocable (holder or chain)	Partial: RFC 7009 plus short lifetimes; no chain concept	Partial: revocation by the servers holding authority,	Yes, at the RS revocation endpoint	Yes, via the Revocation sub-spec	Partial: policy store change by the administrator

			not the holder			
+	Authentication / Proof of Possession	Optional (DPoP, mTLS); bearer remains the deployed default	Yes: every request signed, core mechanism	Yes: HTTP signatures or Data Integrity proofs	Yes: signed envelope, nonce, time bounds	No: out of scope; evidence-issuance checks only
+	Privacy of delegation chain	High by absence: no chain exists to disclose	Partial: history held at the PS, disclosure deployment-defined	No: full chain visible to carrier and verifier	No: same	Possible by architecture: carrier sees nothing, verifier sees everything
+	Offline-capable	Partial: issuance online; JWT verification offline	Partial: verification survives on cached keys; obtaining authority online	Yes for delegation and verification	Yes for delegation and verification	Partial: evaluation offline; granting online and administrative

Reading the matrix

Four patterns are worth drawing out, because they organize everything else in this chapter.

The capability family owns the chain rows. On the criteria that concern the mechanics of delegated authority (confused deputy, chainable, cross-organizational, attenuated, self-revocable: rows 2 and 4 through 7), zCaps and UCANs are the only column pair that is Yes across the board, and they achieve it offline and holder-side. Every other column pays for those

rows with a server dependency, a partial verdict, or both. This is the clearest architectural line in the survey, and it is a family property, not a property of either spec individually.

The identity side owns the semantic rows. On accountability (row 1) and policy expression (row 3), the verdicts invert. The systems that know who everyone is (AAuth's authenticated roles, Cedar's typed principals, OAuth clients with asymmetric auth) can distinguish agent from operator natively; the capability family deliberately keeps principals opaque and delegates that distinction to a companion identity layer. Likewise the richest policy expression in the survey (Cedar) sits in the one spec with no delegation mechanics at all, while the specs with delegation mechanics reserve policy extension points and leave them empty.

Chain privacy inverts too, and for a structural reason. The capability family's defining artifact, the portable chain, is exactly what its privacy row suffers from: the carrier and the verifier both see the whole provenance log. The systems with no portable chain get carrier-side privacy for free, at the cost of an all-seeing server or policy store. Neither side offers the checklist's actual wish (principal and verifier know all, intermediate carrier does not) as a mechanism; the closest approximations are workarounds (ephemeral keys) or deployment choices (PS disclosure policy).

No column is complete. Every spec has at least one No or Partial in the required rows 1 through 7. The capability family misses semantics (rows 1 and 3); the server-mediated family misses holder-side and offline mechanics (rows 4, 6, 7 in various ways); the policy engine misses the delegation artifact entirely (rows 2, 4). This is the observation that motivates both the delegation-stack section and the gaps section below.

Server-Mediated vs. Certificate Delegation

AAuth and the certificate capability family aim at the same use case (autonomous agents acting across organizational boundaries under delegated, revocable, auditable authority) and give opposite answers to the central question of where authority lives. The contrast is the sharpest one available in this survey, precisely because so much else about them agrees: both reject bearer tokens for signed requests, both identify agents by key material, both record who delegated what to whom.

- Portability. A certificate capability is self-contained: the chain travels with the invocation and any verifier can check it. AAuth's delegation history lives in the Person Server's Mission Log; verification of standing depends on authenticated interaction with the server that holds the state.

- Offline delegation. The capability family delegates by signing, with no round trip. AAuth has no offline delegation; obtaining and extending authority are online protocol exchanges. This is the efficiency and partition-tolerance line.
- Attenuation. The capability family enforces algebraic, monotonic attenuation at every link: a child provably holds a subset. AAuth deliberately rejects the subset rule for call chaining; the Person Server evaluates each hop against mission context, which its specification argues is more flexible than algebraic rules. This is a genuine philosophical split (constraint by algebra versus constraint by governance), not an omission on either side.
- Revocation locus. In the capability family, anyone in the chain can revoke, and the enforcement point learns of it. In AAuth, the servers holding authority revoke; the holder of a credential cannot act on the credential itself.
- Key lifecycle. AAuth delegations reference stable agent identifiers, with the cnf claim binding tokens to the current key; identity survives key rotation, at the cost of lifecycle questions about delegations that straddle a rotation. The capability family binds delegations to keys directly, and typically sidesteps rotation with short-lived, delegation-specific keys.

For agentic delegation, the trade reduces to this: the certificate model gives stronger guarantees (offline, locally verifiable, algebraically attenuated, holder-revocable) at the price of thin semantics, while the server-mediated model gives richer semantics and governance (missions, consent, authenticated roles) at the price of a live, trusted server on the path of authority.

The Policy Engine as Complement

Cedar is, almost row by row, the complement of the capability family: strongest exactly where zCaps and UCANs are weakest (policy expression, agent-vs-operator modeling, fine-grained revocation, composition of authority) and weakest exactly where they are strongest (no chain, no portable grant, no holder-side anything). Read as a competitor it fails the survey; read as a component it fills the survey's emptiest cell.

The specific fit: both zCaps and UCANs reserve a policy extension point (caveat; pol) and ship no vocabulary for it. Cedar is a mature, analyzable policy language with no carrier. A capability chain whose caveats were expressed in a Cedar-class language would give the family what its chapters identify as their highest-value missing piece, with one property nothing else in the survey offers: because Cedar is non-Turing-complete and formally analyzable, delegation policies could be audited and subset relationships proven offline, at delegation time rather than invocation time. The reverse composition also holds: Cedar deployments that need portable, cross-organizational grants need exactly the carrier the capability family provides.

This suggests the shape of a composed delegation stack, assembled from surveyed parts:

1. Entry point (identity layer): an identity system (OIDC, Verifiable Credentials) makes the human or organizational judgement call and binds operator to agent; the zCap specification itself recommends this split.
2. Authority flow (carrier layer): a certificate capability chain carries the delegated, attenuated authority across hops and organizations, offline and locally verifiable.
3. Restrictions (policy layer): caveats within the chain expressed in a standardized, analyzable policy language, evaluated at each hop and provable between hops.
4. Presentation (possession layer): signed invocation of the transport in use (HTTP Message Signatures, Data Integrity proofs, DPoP-style binding), with keys held outside the agent per the Introduction's requirements.

No surveyed spec provides more than two adjacent layers of this stack, and no pair of them currently interoperates by standard. That observation leads directly to the gaps.

Gaps

Reading down the merged matrix, the recurring Partial and No cells cluster into six gaps that no surveyed specification fills. These are candidates for working group attention, ordered roughly by how directly they block agentic delegation.

1. A standard policy and caveat vocabulary for delegation control. Both certificate capability specs have the extension point and neither ships vocabulary; AAuth places policy outside the protocol; OAuth scopes are opaque strings. The concrete missing terms are the checklist's own examples: undelegatable, re-delegation constraints, delegation depth. Cedar demonstrates that such a vocabulary can be expressive and formally analyzable at once. This is the gap where existing pieces come closest to composing, and arguably the highest-value item.
2. An advisory tier. No surveyed system can express a request whose violation is not refused ("please do not re-delegate outside the org"): every evaluation model in the survey is binary enforce/reject. Advisory semantics matter for agentic delegation because agents cross governance boundaries where hard enforcement is impossible but recorded intent still has value (for audit, for liability, for downstream policy).
3. Agent-vs-operator semantics and identity binding. The capability family cannot say which principal is the accountable legal entity; the identity-centric systems can, but only inside their own trust domain. What is missing is a standardized companion mechanism (the VC-based binding both capability chapters gesture at) so that accountability to an operator survives cross-organizational, multi-hop delegation without collapsing back into identity-based authorization.
4. Delegation chain privacy. The checklist's formulation (principal and verifier know the full story, the intermediate carrier does not) is offered by no spec as a mechanism. The capability family discloses everything to everyone on the path; the server-mediated and

policy families relocate the problem to an all-seeing server. Selective disclosure or blinding of chain interiors is an open design problem.

5. Offline and decentralized revocation. Every surveyed revocation mechanism is state at an enforcement point that must be reached: an RS endpoint, a server's grant, a policy store. For the local-machine and LAN agentic use cases the checklist highlights, revocation semantics under partition are unspecified everywhere. UCAN's open dissemination model (gossip, DHT) is the closest starting point, but its guarantees are undefined.
6. Cross-organizational policy exchange. Even where evaluation is local, policy semantics are unilateral: Cedar's store belongs to one verifier, an AS's scope definitions to one provider, a PS's mission policy to one server. There is no standard by which one organization hands another a policy fragment with agreed meaning. This is the cross-organizational criterion's unmet half, and it compounds gap 1: a shared caveat vocabulary is also the natural interchange format.

Conclusion

The survey's answer to "which spec should agentic delegation use" is that the question is mis-posed. The certificate capability family supplies the only viable backbone for the authority flow itself: it alone meets the chain criteria, offline and holder-side, and it alone is immune to the confused deputy by construction. But it is a backbone, not a solution; its own chapters identify the missing semantics, and those semantics are exactly what the identity-centric and policy systems in this survey already know how to express. The near-term work is therefore not a sixth competing spec but composition: a standardized policy vocabulary that can ride in a capability chain, a standardized identity binding at the chain's root, and revocation and privacy mechanisms designed for the partitioned, multi-organizational environments agents actually operate in. The gaps list above is, in effect, the specification outline for that work.

input Chapter: OAuth2

Chapter: OAuth

Hardening, Consolidation, and the Push Toward Autonomous Authorization

---

SCOPE

This chapter covers what is actually deployed today — OAuth 2.0 (RFC 6749/6750), as hardened by RFC 9700 (2025 Security Best Current Practice) — then traces the evolution toward OAuth 2.1 and, separately, the independent AAuth draft.

Each section is tagged as one of:

- Deployed standard
- Adjacent RFC/profile
- OAuth 2.1 — consolidation draft
- AAuth — independent draft

This makes it clear what you'd encounter talking to Google, Okta, Auth0, GitHub, or a typical enterprise IdP today, versus what's still in motion.

Overview

OAuth (Open Authorization) is an open standard for **access delegation**. It allows users to grant third-party applications secure, scoped access to their data hosted on another service without ever handing over their passwords. Much like a "valet key" for a car—which lets a valet drive and park the vehicle but prevents them from unlocking the glovebox—OAuth uses limited-scope **Access Tokens** rather than shared credentials to protect user data.

The standard running in production today — what every major identity provider implements to achieve this — is OAuth 2.0: RFC 6749 (the core authorization framework) and RFC 6750 (bearer token usage), both from 2012. OAuth 2.0 is a framework, not a fixed protocol: it deliberately leaves many security-relevant decisions to implementers. This flexibility is precisely why a decade of real-world attacks produced a long list of follow-on RFCs tightening specific gaps (PKCE, JWT access tokens, token exchange, sender-constraining, and so on).

In January 2025, the IETF published RFC 9700, "Best Current Practice for OAuth 2.0 Security," which consolidates that decade of lessons into concrete guidance: don't use the Implicit grant, don't use Resource Owner Password Credentials, always use PKCE, match redirect URIs exactly, prefer sender-constrained tokens. RFC 9700 is guidance on top of RFC 6749/6750 — it doesn't replace them.

OAuth 2.1 (draft-ietf-oauth-v2-1) is the next step in that consolidation: an attempt to fold RFC 9700's hardening directly into a single normative document that obsoletes RFC 6749/6750. As

of this writing, it is still an Internet-Draft — "OAuth 2.1" is best understood as the direction the ecosystem is consolidating toward, not a separately deployed protocol version.

Separately, AAuth (draft-hardt-oauth-aauth-protocol) is an independent, early-stage draft built specifically for agent-to-resource access, designed to coexist with OAuth/OIDC — not extend either.

Part 1: What's Actually Deployed — OAuth 2.0 + RFC 9700

The Core Framework

Deployed standard	RFC 6749 defines the four classic grant types (Authorization Code, Implicit, Resource Owner Password Credentials, Client Credentials) plus Refresh Tokens. RFC 6750 defines how bearer tokens are presented to a resource server. This is the substrate nearly every production OAuth deployment is built on.

RFC 9700: The Hardening Layer in Force Today

Deployed standard / BCP	RFC 9700 is the document that tells you what a secure OAuth 2.0 deployment looks like right now.

RFC 9700 mandates the following:

- Don't use the Implicit grant (response_type=token) — tokens returned in a URL fragment are exposed to browser history, referrer leakage, and any script on the page.
- Don't use Resource Owner Password Credentials — it trains users to type credentials into third-party UIs and gives the client raw password access.
- Use PKCE (RFC 7636) on the Authorization Code flow for all clients, not just public ones — it closes authorization-code-interception attacks that affect confidential clients too.
- Match redirect URIs exactly; no substring or wildcard matching.
- Prefer sender-constrained tokens (DPoP or mutual TLS) over plain bearer tokens where feasible.
- Use state for CSRF protection and PKCE's code_verifier together — they protect against different attacks.

None of this requires "OAuth 2.1" as a separate thing to implement. OAuth 2.1's contribution is mostly editorial consolidation: taking these BCP recommendations and making the safe behaviors the only specified behaviors.

Client Authentication: Required vs. Common Practice

Deployed standard / Adjacent RFC	RFC 6749 only requires that confidential clients authenticate to the token endpoint somehow. client_secret_basic and client_secret_post are the baseline methods, widely used today.

Asymmetric client authentication has become common best practice for higher-assurance deployments:

- private_key_jwt (RFC 7523) — the client signs an assertion with its own private key instead of sending a shared secret.
- Mutual TLS client authentication (RFC 8705) — the client authenticates via its TLS certificate.

These are additions layered onto OAuth 2.0 — not something RFC 9700 or OAuth 2.1 mandates outright.

Proof of Possession: DPoP and mTLS

Adjacent RFC	By default, OAuth 2.0 bearer tokens (RFC 6750) are bearer instruments. Whoever holds the token can use it.

Two specs address this:

- DPoP (RFC 9449) — the client generates a key pair and signs a proof bound to each request's HTTP method and URI. A stolen token is useless without the matching key.
- Mutual TLS (RFC 8705) — binds the token to the client's TLS certificate.

RFC 9700 recommends sender-constraining; it doesn't require it, and most OAuth 2.0 deployments still issue plain bearer access tokens.

Token Format: Opaque vs. Self-Contained

Deployed standard / Adjacent RFC	OAuth access tokens can be opaque (validated via token introspection, RFC 7662) or self-contained JWTs (verifiable offline per the JWT Access Token profile, RFC 9068). The choice is left to implementers by RFC 6749.

Revocation

Deployed standard / Adjacent RFC	RFC 7009 lets a client or resource owner request explicit revocation. Short-lived access tokens provide a passive backstop. OAuth 2.0 has no native notion of delegation-chain revocation — it has no native delegation chain in the first place.

Part 2: OAuth 2.1 — Where the Consolidation Is Heading

OAuth 2.1 — consolidation draft

OAuth 2.1 (draft-ietf-oauth-v2-1) takes everything in Part 1's hardening layer and makes it the text of the spec itself.

Key changes in OAuth 2.1 vs. today's hardened OAuth 2.0:

- Implicit and ROPC grants are removed from the spec text, not just discouraged.
- PKCE is required, not recommended, for every client on the Authorization Code flow.
- Redirect URI exact-matching is specified, not left as guidance.
- Sender-constrained tokens (DPoP/mTLS) remain a SHOULD, not a MUST. OAuth 2.1 does not mandate proof-of-possession tokens.
- Client secrets are still supported. The draft explicitly requires AS support for clients sending a plain client_secret. Asymmetric client authentication is not the OAuth 2.1 baseline.

If you're running a well-hardened, RFC 9700-compliant OAuth 2.0 deployment today, you are very close to "OAuth 2.1 compliant" already. OAuth 2.1's main value is editorial — a single document new implementers can build against without independently discovering a decade of separate BCP RFCs.

Part 3: AAuth — A Separate, Independent Draft for Agent-to-Resource Access

AAuth — independent draft

Caveat: this is a single-author Internet-Draft, not an IETF working-group product. It has already been renamed and restructured more than once. Treat the below as a snapshot of a moving target.

Why It Exists

AAuth (draft-hardt-oauth-aauth-protocol) was created by an OAuth 2.0/2.1 co-author who concluded that OAuth, in any version, is a poor fit for agent-to-resource access:

- Agents need authorization decisions made mid-task against resources they've never interacted with before.
- Agents have no durable client_id that travels across organizations the way OAuth assumes.
- Copying long-lived API keys into autonomous workloads recreates the shared-secret leak surface OAuth 2.0 and 2.1 are trying to move away from.

AAuth is explicitly designed to coexist with OAuth 2.0 and OpenID Connect, not extend either one.

Architecture

Roles:

- Agent
- Agent Provider (AP)
- Person Server (PS)
- Resource

Token types:

- agent_token — establishes the agent's identity
- resource_token — describes the access a resource needs
- auth_token — grants access to a specific resource, where applicable

Core mechanism: signed HTTP requests using HTTP Message Signatures, not bearer tokens trusted on presentation. The simplest mode requires no token issuance flow at all.

Four Resource Access Modes

AAuth defines four modes, increasing in complexity:

- Identity-based access — the agent signs requests with its agent_token; the resource verifies the signature and applies its own access policy. No authorization flow, no additional tokens. This replaces API keys with cryptographic identity outright.
- Resource-managed access (two-party) — the resource handles authorization itself, optionally returning an opaque token for subsequent calls.
- PS-asserted access (three-party) — a Person Server vouches for claims about the person on whose behalf the agent acts, with a consent step and an auth_token issued on approval.
- Federated access (four-party) — extends the three-party model across an additional trust boundary.

Key discovery: agents and resources discover signing keys via well-known metadata and JWKS-style endpoints — similar in spirit to OAuth's JWKS discovery, but used for HTTP-signature key lookup, not bearer-token verification.

Part 4: Scorecard — OAuth 2.0, AAuth, and UCAN/zCAP-LD

Requirement	OAuth 2.0 (RFC 9700)	AAuth	UCAN / zCAP-LD
Accountable (agent vs. principal)	Yes (with asymmetric auth); weaker with shared client_secret	Yes — every agent has its own signing key	Yes — capabilities are signed by a specific issuer
Resistant to confused deputy			
Represent policies	Partial — scopes, resource indicators; expressiveness depends on extensions deployed	Partial — policy logic lives with the resource or Person Server	Yes — caveats/conditions embedded directly in the capability
Chainable (multi-hop delegation)	No — single-hop; multi-hop needs online Token Exchange (RFC 8693)	No native multi-hop offline chain in the core spec	Yes — native, offline-verifiable multi-hop delegation
Cross-org / locally verifiable	Conditional — true for JWT access tokens; false for opaque tokens	Yes for the signature step; multi-party modes need a live round trip on first use	Yes — needs only the chain and relevant public keys
Attenuated	Yes, via the AS (scope narrowing, Token Exchange) — not unilaterally by the client	Resource-managed/PS-asserted modes can scope tokens narrowly	Yes — a holder can mint a narrower sub-capability offline
Self-revocable	Partial — RFC 7009 + short lifetimes; not a delegation-chain concept	Depends on mode; identity-based access has no token to revoke, only key rotation	Generally needs an external registry; chains are otherwise fire-and-forget
Composable	No native cryptographic composition	No documented composition primitive	Partial, implementation-dependent
Proof of Possession	Optional — DPoP/mTLS available, not required; most still issue plain bearer tokens	Yes, inherently — every request is signed by default	Out of scope of core data model; typically layered on via transport
Privacy of delegation chain	High for opaque/minimally-scoped tokens; lower for JWTs with identity claims	High — resource learns only agent identity and vouched person claims	Lower by default — verifier may need to inspect the full chain
Offline-capable	Partial — issuance always online; verification offline only for self-contained JWTs	Partial — identity-based mode fully offline-verifiable; multi-party modes need one online round trip	Partial — delegation offline by design, revocation checking typically isn't

---

Closing Note

What you'll actually meet in production today is OAuth 2.0, hardened to RFC 9700's recommendations, with client secrets still alive and well at most token endpoints and bearer tokens still the default token type. OAuth 2.1 is where that hardening is headed as a single

normative document, but it changes less than its name suggests — no mandated PoP, no eliminated client secrets.

AAuth, meanwhile, isn't a future version of OAuth at all; it's a parallel, signature-based protocol built because its author judged the OAuth model itself a poor match for autonomous agents. And UCAN/zCAP-LD remain the only one of the three with native, offline, multi-hop delegation — the clearest architectural line separating capability systems from both the OAuth lineage and AAuth.

input: AAuth

Chapter: AAuth

Overview

AAuth is an emerging authorization architecture designed for authenticated interactions among people, AI agents, resources, Person Servers (PS), and Agent Providers (AP). Rather than treating software agents as extensions of end users, AAuth introduces authenticated agents as first-class participants capable of invoking protected resources, receiving asynchronous events, and operating across organizational boundaries.

Unlike capability-based delegation systems such as UCAN and zCaps, AAuth does not represent delegated authority as a self-contained, cryptographically attenuated artifact carried entirely within a delegated credential. Instead, delegation is server-mediated. Authority is anchored within the Person Server, with delegation history represented through the act claim, Mission References, and the Mission Log maintained by the Person Server. AAuth therefore defines explicit delegation mechanics while intentionally separating delegation state from portable capability objects.

AAuth is evaluated here as a server-mediated delegation architecture. It authenticates participants, establishes trusted execution between them, records delegation history, and supports authenticated event-driven execution. Questions concerning delegation semantics, attenuation policy, authority evolution, and the policy logic behind execution-time decisions remain outside the protocol itself and are expected to be supplied by complementary architectural layers.

Three reference points are used:

• draft-hardt-oauth-aauth-protocol-10 (6 August 2026), together with draft-hardt-httpbis-signature-key-08, the companion specification to which AAuth defers key conveyance, discovery, and algorithm determination;

• the AAuth Events draft, which extends the architecture with authenticated asynchronous event delivery between resources and agents through Agent Providers; and

• public implementation discussions describing the intended architecture and deployment model.

Where protocol capabilities and implementation practices differ, this distinction is noted throughout the evaluation.

A note on method: this chapter was drafted with AI research collaborators under a gated review process, with final editorial authority held by the author. Load-bearing claims were verified directly against the AAuth specification repository rather than model recall — including the token-exchange question, the act claim's provenance, credential key-binding, and the Appendix B.3.7 attenuation rationale.

SCORECARD

1. Accountable (agent vs. principal/operator)

Verdict: Yes

AAuth distinguishes authenticated people, agents, Person Servers, Agent Providers, and resources. Delegation history records acting parties through the act claim.

2. Resistant to confused deputy (no ambient authority)

Verdict: Partial

The resource-managed, PS-asserted, and federated modes bind an authorization to a designated resource and scope through the resource token, combining designation with authorization. The OPTIONAL account parameter added in -10 narrows that binding further, naming which account at the resource the authorization covers and propagating through the resource token into the auth token. Identity-based access does not do this: the resource authorizes on the agent's identity alone and applies its own access control, which is ambient authority.

3. Represent authorization policies

Verdict: Partial

Authorization context and Mission References are represented, but expressive delegation policy—for example, undelegatable authority, advisory policy, or attenuation constraints—is intentionally outside the protocol.

4. Chainable

Verdict: Yes

Delegation chains are represented through the act claim and the Mission Log. Chains are server-mediated rather than portable capability artifacts. Sub-agent nesting is single-level; deeper workflows use chained top-level agents, each an independent principal holding its own grant.

5. Cross-organizational / locally verifiable

Verdict: Yes

Designed for authenticated interactions across organizational boundaries while preserving local trust relationships.

6. Attenuated

Verdict: Partial — differs by mode

Sub-agent authorization supports attenuation: every request passes through the parent, which can refuse, attenuate, or rate-limit. Call chaining does not — downstream authorization is intentionally not required to be a subset of upstream scope. Cryptographically enforced attenuation is not intrinsic to the protocol in either case.

7. Self-revocable

Verdict: Partial

Token revocation, mission revocation, and propagation through a parent's grant are all specified. Revocation is exercised by the servers holding authority rather than by a holder acting on a credential in hand, and mission state administration beyond completion is deferred to a companion specification.

Authentication / Proof of Possession

Verdict: Yes

Core capability of the protocol. -10 requires a fully-specified alg identifier, recommends Ed25519, and prohibits none, symmetric algorithms, and the polymorphic EdDSA identifier that RFC 9864 deprecated.

Privacy of Delegation Chain

Verdict: Partial

Delegation history is maintained by the Person Server rather than being universally disclosed through portable credentials. The protocol records delegation continuity while allowing deployment architectures to determine how much of that history is exposed during execution.

Offline Capable

Verdict: Partial

Implementations MUST cache JWKS and SHOULD continue verifying against cached keys when a fetch fails, bounded by a cache lifetime of at most 24 hours, so token verification survives temporary loss of contact with an issuer. Obtaining authority remains online: AAuth's server-mediated model provides no offline delegation, and initial key discovery requires reachable metadata.

DETAILED EVALUATION

1. Accountable (agent vs. principal/operator)

AAuth explicitly distinguishes among authenticated people, AI agents, Person Servers, Agent Providers, and protected resources. Rather than treating software agents merely as extensions of user sessions, the protocol gives each participant its own authenticated identity and architectural role.

Delegated actions are represented through the act claim, allowing delegation history to be maintained across direct delegation, sub-agent delegation, and chained execution. Mission References further associate requests with the governing mission maintained by the Person Server.

Unlike certificate capability-based systems, accountability derives from authenticated protocol exchanges combined with server-maintained mission history rather than from cryptographically self-contained delegation credentials.

The specification also permits a resource to authorize an agent based solely on its identity, without interaction or mission context; working-group discussion has flagged that identity-only authorization reintroduces confused-deputy exposure, since the execution context rather than the requester's identity is what validates an action.

2. Resistant to confused deputy (no ambient authority)

The confused deputy problem arises where a party exercises its own permissions on a resource designated by someone else. The defence is to combine designation with authorization, so that what may be done is bound to what it may be done to.

In the resource-managed, PS-asserted, and federated modes AAuth does this. The agent obtains a resource token naming the resource and scope; the auth token issued against it carries that binding, and the resource enforces it. The OPTIONAL account parameter, added in -10, narrows the binding further: it names which account at the resource the authorization is for, drawn from the resource's own namespace, and is echoed through the resource token into the auth token, so an auth token for one account grants nothing at another.

Identity-based access is the exception, and a deliberate one. The agent signs requests with its agent token, and the resource applies its own access control based on who the agent is. There is no authorization flow and no designation carried with the request. The specification presents this as a replacement for API keys, which it is; but authority derived from identity alone is ambient, and an agent acting on a request supplied by another party has no way to signal that the request is not its own.

3. Ability to Represent Authorization Policies

AAuth represents authorization context through missions, authenticated requests, and protocol-defined execution relationships. Mission References bind requests to an existing mission while allowing authorization policy to remain under the control of the Person Server.

The protocol intentionally does not standardize a rich delegation policy language. Concepts such as undelegatable permissions, advisory delegation preferences, attenuation constraints, or policy composition remain responsibilities of external policy engines or governance frameworks.

This separation allows AAuth to remain focused on authenticated execution while supporting a wide variety of higher-level authorization models.

4. Chainable

AAuth provides explicit delegation mechanics through the act claim and the Mission Log. These mechanisms record delegation history across direct delegation, chained delegation, and sub-agent authorization.

Unlike certificate capability-based systems, delegation history is maintained by the Person Server rather than embedded entirely within a portable delegation credential. Verification therefore depends upon authenticated interaction with the server maintaining mission state rather than validating an independently portable capability chain.

Accordingly, AAuth supports delegation chains, but those chains are server-mediated rather than self-contained cryptographic artifacts.

Sub-agent nesting is limited to a single level: a sub-agent must not have sub-agents of its own, and the specification enforces this from both directions. Deeper workflows are carried instead by call chaining, where each hop is an independent top-level agent holding its own grant rather than a recursive sub-agent — which is also why upstream-subset rules do not apply to it. What the specification defers to a companion document is mission state administration beyond completion: revocation state transitions, delegation-tree queries, and administrative interfaces. Delegation-chain lifecycle is not deferred; sub-agent revocation propagates through the parent's grant.

5. Cross-Organizational / Locally Verifiable

Supporting authenticated interactions across organizational boundaries is one of AAuth's principal architectural objectives.

Person Servers, Agent Providers, agents, and protected resources may participate across independent trust domains while preserving authenticated relationships between participants. Authentication evidence can therefore span organizations without requiring centralized identity management.

Trust remains anchored in authenticated protocol interactions and configured trust relationships rather than portable authorization credentials.

From a capability-based perspective, the need for Access Server federation may indicate that authority remains server-mediated rather than fully represented by the delegation credential

itself. If the delegation credential carried sufficient authority for cross-domain verification, no ongoing relationship between authorization servers would be required.

6. Attenuated

Attenuation in AAuth differs by delegation mode. Sub-agent authorization supports it directly: every sub-agent request passes through the parent, which can refuse, attenuate, or rate-limit. Call chaining does not, and does so by design — downstream authorization is intentionally not required to be a subset of upstream scope, and downstream scope is constrained by the downstream resource's own policy together with Person Server evaluation against mission context.

Unlike UCAN or zCaps, attenuation is not represented as a cryptographically enforced property of a portable delegation artifact. Instead, attenuation depends upon server-managed mission state and policy evaluation.

The protocol therefore leaves attenuation semantics outside its definition, supplying contextual evaluation in place of algebraic constraint.

This is deliberate design rather than omission. AAuth does not adopt the RFC 8693 token-exchange model for attenuation (it borrows only that RFC's act claim structure for representing delegation chains), and Appendix B.3.7 states the rationale directly: downstream scope is intentionally not required to be a subset of upstream scope, because "the PS evaluates each hop against the mission context, providing governance-based constraints... more flexible than algebraic attenuation rules." The one place attenuation exists within an agent's own purview is sub-agent authorization, where the parent "can refuse, attenuate, or rate-limit" the sub-agent's authority.

7. Self-Revocable

AAuth includes mechanisms for token revocation and lifecycle management.

-10 specifies named revocation scenarios: a Person Server revoking an auth token it issued or provided, an Access Server revoking one it issued, an agent provider revoking an agent token, and a Person Server revoking a mission, after which subsequent token requests referencing that mission are denied. Revocation also propagates structurally — revoking a parent's grant causes the next sub-agent authorization to fail, while tokens already issued expire within the hour.

The specification is candid about the limits. Verifying an auth token does not consult its issuer, so nothing in the verification path reports that a token has been revoked; a party no revocation request reaches is bounded by token lifetime alone. What remains unspecified is mission state administration beyond completion, deferred to a companion specification. Revocation is therefore exercised by the servers holding authority rather than by a holder acting on a credential in hand.

SUMMARY

AAuth represents a distinct approach to delegated authorization for AI systems. Rather than expressing delegated authority as portable cryptographic capabilities, it anchors authority within the Person Server and maintains delegation continuity through authenticated protocol exchanges, mission state, and recorded delegation history. In the taxonomy of capability models, AAuth authorizations are neither bearer capabilities nor certificate capabilities: they are server-mediated proof-of-possession credentials, and the specification assigns multi-hop safety to mission-context evaluation rather than to algebraic attenuation, expecting complementary layers to supply the latter where required. Delegation in AAuth references stable agent identifiers rather than individual keys. The cnf claim binds each authorization token to the agent's current signing key while the identifier persists across key rotation. This reduces coupling between delegation history and key lifetime, but introduces lifecycle questions concerning delegation validity across key rotations — whether a delegation was issued before or after a rotation, and how invocations arriving after should be treated. Capability systems using delegation-specific or one-time keys largely avoid this issue, since delegated credentials typically expire before key rotation becomes relevant.

Its strengths lie in authenticated interoperability, cross-organizational execution, event-driven interaction, and maintaining coherent delegation history throughout mission execution. Unlike capability-based systems, authority remains associated with server-managed mission state rather than being embodied entirely within portable delegation artifacts.

This architectural choice intentionally separates authenticated execution from higher-level governance concerns. Authentication and delegation establish who is requesting execution and under what delegated authority. The Permission Endpoint now provides a normative execution-time decision surface: an agent may ask whether a specific action is permitted and receive granted or denied, recordable in the mission log. What remains outside the protocol's scope is the policy logic that produces that answer, together with attenuation policy, authority evolution, mission governance, and dynamic authority state.

Accordingly, AAuth is best understood as a server-mediated delegation and execution framework. It provides authenticated execution, delegation continuity, and mission coordination while allowing complementary policy and governance architectures to determine whether delegated authority remains valid as mission context, organizational policy, and authority state evolve.

input: zcaps

Chapter: zCaps (Authorization Capabilities)

Overview

Authorization Capabilities – “zCaps”, standardized (in draft) as Authorization Capabilities v0.3 (ZCAPs) – are an implementation of the object-capability model for the web. A zCap is a structured, cryptographically-signed access token that answers “which **agent** (`controller`) **can** do **what** (`allowedAction`) **with** which resource (`invocationTarget`), **given** what restrictions (caveats / attenuation)”.

Unlike Access Control List (ACL) systems, which decide access by *identity* (“who are you?”), zCaps decide access by *possession* (“do you hold a valid capability, and can you prove control of its key?”). This is the central design choice that shapes how zCaps score against the matrix: they are strong on the delegation-chain mechanics (chaining, attenuation, accountability of each delegation edge, local verification), and less focused on *identity semantics* or about *combining* authority from multiple independent sources.

Two reference points are used in this evaluation:

- the ZCAP v0.3 specification (the data model), and
- the zCap Developer’s Guide, which documents the *deployed* state of the art, which in several respects has diverged from the drafted spec (most notably: typed `caveat` objects are largely unused in production, replaced by URL-suffix attenuation, `allowedAction`, and `expires`).

Where the deployed behavior and the spec text differ, this is called out explicitly, since it materially affects the score on policy expressiveness.

Scorecard

#	Requirement	Verdict	Notes
1	**Accountable** (agent vs. principal/operator)	Partial	Every delegation edge is signed and thus attributable to a responsible delegator, but the data model has no semantics distinguishing “agent” from “legal-entity operator” – both are opaque DIDs.
	Resistant to confused deputy	Yes	As a capability system, resistant to confused deputy by definition
2	**Represent authorization policies** (embed/link; “undelegatable”, re-delegation rules, advisory “please”)	Partial	Generic `caveat` mechanism exists in the spec but is largely unused in deployment. Note: no standard vocabulary for “no further delegation”, since that's an anti-pattern. Also, currently no re-delegation policy, or *advisory* (non-enforced) rules.
3	**Chainable**	Yes	This is the core feature: `parentCapability` + `capabilityChain` proof chains.
4	**Cross-organizational / locally verifiable**	Yes	Verification is cryptographic and local; the full chain is carried in the invocation; implicit preference is for identifiers that resolve to keys, and without accounts on each other’s IAM system/registry/etc., such as DIDs.

5	**Attenuated**	Yes*	Currently (as implemented), attenuation is *implicit:* only appears as `allowedAction` subset, more-restrictive `expires`, and URL path/query suffix narrowing of `invocationTarget`, all monotonically enforced by the verifier. There is an explicit **caveat** property that's reserved, but is unused.
6	**Self-revocable** (holder or anyone in chain)	Yes (at RS)	Any controller in the chain may revoke their zCap (which revokes everything downstream) by posting the zCap to the RS revocation endpoint; out-of-band, RS-enforced, so online-only.
+	Authentication / Proof of Possession	Yes	HTTP Message Signatures (Cavage/RFC 9421) or a DataIntegrity `capabilityInvocation` proof; replay-resistance is offered through short expirations.
+	Privacy of delegation chain	No	The full chain is submitted to the verifier and is visible to the carrying agent; no selective disclosure of intermediate delegates.
+	Offline-capable	Possible	Delegation and chain verification can be offline; *invocation* and *revocation* depend on the RS (if the RS is offline-capable, then possible).

(zCaps) Detailed Evaluation

1. Accountable – agent vs. principal/operator

Can zCaps differentiate between the agent and its principal operator (a legal entity)? zCaps satisfy the *accountability* half of this well and the *differentiation* half only implicitly.

The agent (actor) is named directly: the `controller` field of each zCap is the DID of the key-holder authorized to invoke or further delegate it. Accountability across the chain is a structural guarantee: every delegated zCap carries a `capabilityDelegation` proof signed by a controller of its `parentCapability`.
So, for every edge in the delegation graph there is a cryptographically-attributable, non-repudiable "responsible party" – exactly the property the matrix calls for ("each delegator must be able to assign a responsible party for each delegation"). Walking the chain back to the root recovers the complete provenance of any authority.

What zCaps do not provide is a semantic distinction between "this DID is an autonomous agent" and "this DID is the legal entity / operator that stands behind it". A `controller` DID is opaque; the root-zCap controller is effectively the operator/admin only by convention (it sits at the root of trust the RS dereferences).

Binding key identifiers to real-world legal entities

Binding a DID (or other key identifier) to a real-world legal entity is explicitly pushed out of scope. For one, it's not always desirable or possible -- many agents or other delegates do not have their own identity, or doing so would be a privacy violation.

If key identifier to legal entity mapping is needed, it can be achieved with:

- issuer registries
- with Verifiable Credentials (the spec's "Relationship to Verifiable Credentials" section recommends exactly this split: use VCs for the human/identity judgement call at the entry point (possibly to *get* the authorization), and use zCaps for the flow of authority thereafter). So, an auditor can always identify *which key* delegated *which authority to which other key*, but the operator-vs-agent distinction intentionally lives in a companion identity layer, not in the zCap data model.

Note that auditability requirements can be fulfilled with local pairwise identifiers only, and that global identities are not necessary.

2. Ability to represent authorization policies

This is the least emphasized area for zCaps as currently deployed (the current use cases they're employed for have not called for expressing policy in the data model).

The spec defines a general `caveat` property: typed restriction objects (the examples are `ValidWhileTrue` and `DriveNoMoreThan`) that are inherited down the chain and whose meaning "must be understood between the entity adding a caveat and the target evaluating it." There is no standard caveat vocabulary; currently attenuation is expressed implicitly (attenuation via shortening expiration, invocationTarget url, or allowed actions list). The Developer's Guide notes that in production no caveat notation is currently used – attenuation is expressed instead through `allowedAction`, `expires`, and URL-suffix narrowing.

Against the specific policy examples in the matrix:

- **"Do not delegate further" / "this permission is undelegatable":** not natively expressible in deployed zCaps. There is no depth-limit or non-delegation flag in the data model. The only delegation-depth control is a *verifier-global* policy (a verifier MUST cap chain length, SHOULD cap at 10) – that is the RS's blanket rule, not a per-capability instruction a delegator can set.
- **Re-delegation policies:** likewise not standardized; would need custom caveats.
- **Advisory "please" rules (Alan's point – desired re-delegation rules that are *requested* but whose violation is not refused):** zCaps have no notion of advisory, non-enforced policy. Caveats are evaluated at invocation and a failed caveat invalidates the invocation. The model is binary enforce/reject, with no "honor-if-possible" tier.
- **"Who gets to know what is a governance topic":** consistent with the OCap philosophy, zCaps deliberately encode *authority*, not *knowledge/identity policy*; governance over disclosure is out of scope.

So the data model *can* carry policy via the extensible `caveat` slot, but it ships with no shared vocabulary for the policies the matrix cares about, and current deployments don't use it at all. This is arguably the highest-value area for the "future work" of the spec if it is to serve agentic delegation.

3. Chainable

This is the headline strength of zCaps and the reason the model exists. Delegation is expressed by `parentCapability` (pointing up at the parent zCap or the root)

plus a `capabilityDelegation` proof whose `capabilityChain` enumerates the ancestry. The root of authority is the resource itself (the root zCap, controlled by the operator); authority then flows root to agent to sub-agent. Each link can be created *offline* simply by signing a new child zCap with a key the parent authorized – no involvement of the resource server is required to delegate.

This directly satisfies both senses the matrix asks for: principal/operator to agent (root to first delegated zCap), and agent to further subsystems/other agents (delegated to delegated). The "valet" example in the spec (car to Alyssa to Ben to Lem) is precisely a multi-hop agent-to-agent delegation chain.

4. Cross-organizational / locally verifiable

Strong, and again by design. Verification is purely cryptographic and local to the verifier:

- A delegated zCap must carry its entire chain in the invocation; the spec mandates that a verifier must not be *required* to make network requests or database queries to dereference the chain. The verifier traverses from the root down, checking each `capabilityDelegation` proof against the controller authorized by the step above, and checking that each step only *narrows* authority.
- Controllers are DID-based key identifiers, which are decentralized – no party needs an account on another party's IAM system. Two organizations can delegate to each other purely by exchanging signed zCaps.
- The **root of trust is the RS**: a verifier dynamically dereferences the root zCap's `controller` from its own store for the protected resource.
- **DID resolution and revocation checks** may still touch the network depending on the DID method (e.g., `did:key` is self-contained and offline; `did:web` requires an HTTP fetch) and the RS's revocation store. Chain *structure* verification is offline; freshness checks are not.

Net: the data model is independently and locally verifiable across organizational boundaries.

5. Attenuated

zCaps enforce monotonic attenuation along four axes, all checked by the verifier:

- **Actions:** `allowedAction` in a child MUST NOT be less restrictive than in the parent's.
- **Time:** `expires` MUST NOT be later than the parent's (and every delegated zCap MUST have an expiry).
- **Resource:** `invocationTarget` may be narrowed by appending a URL path or query suffix (`/bars/123` to `/bars/123/bazzes/456?day=tuesday`); a child's target MUST be the parent's target or a prefix-extension of it.
- **Caveats:** the caveat property is reserved in vocabulary, but not currently used by implementations.

A delegator can therefore hand out no more than they hold. The main limitations are expressiveness, not capability: attenuation of the resource is restricted to URL-suffix shaping (fine for hierarchical REST resources, awkward for non-hierarchical scoping), and – per the Developer's Guide – a single zCap is limited to **one** `invocationTarget`, with multiple target/action combinations listed as future work. So, yes, with the note that richer attenuation is a known gap.

6. Self-revocable

The matrix requires that the holder, or *someone in the delegation chain*, be able to revoke (modulo network partitions). zCaps meet this: any controller appearing in a delegated zCap's chain may revoke it by POSTing the zCap to the RS's revocation endpoint (the spec sketches a `/zcaps/revocations` subpath whose root zCap is controlled by *all* controllers in the chain, precisely so that any ancestor can revoke). Mandatory `expires` on every delegated zCap bounds how long a revocation must be remembered (and bounds damage from a lost-but-unrevoked zCap).

The important characteristics: revocation is **out-of-band and RS-enforced** – there is no decentralized/cryptographic revocation embedded in the token itself; it is a state change at the resource server, which is the single enforcement point. That makes it **online-only** (you must reach the RS), which is consistent with the matrix's "modulo network partitions" caveat but worth stating plainly for agentic use cases that may want offline revocation semantics. **Yes (RS-dependent).**

Nice-to-haves

Authentication / Proof of Possession

Possession alone is never sufficient; invocation requires proving control of the `controller` key. Two binding mechanisms exist: an HTTP Message Signature (deployments use Cavage HTTP Signatures Draft 12 today, migrating to RFC 9421) over a `Capability-Invocation` header (plus `Digest`/`Content-Digest` for bodies), or a Data Integrity `capabilityInvocation` proof attached to a request document. Because the request itself is signed (with `created`/`expires` `id`), intercepted invocations cannot be replayed without the secret key – analogous to DPoP.

Privacy of the delegation chain (No)

The full delegation chain MUST be submitted to the verifier on invocation, and is carried by (and therefore visible to) the invoking agent itself. There is no current mechanism for selective disclosure, encryption, or blinding of intermediate delegates: the principal, the verifier, *and* the token-carrying agent all see the whole chain. (Delegated-zCap `id`s use `urn:uuid` to reduce *correlation* of identifiers, but that does not hide the chain contents.) While there is no explicit blinding mechanism (no way to hide the key id), you can achieve most of the desired effect by using ephemeral one-off keys.

Offline-capable

The offline story splits by lifecycle stage:

- **Root creation and delegation: fully offline.** A root zCap is a deterministic template; delegating is just signing a child zCap with an authorized key. Crucially, *you do not contact the resource server to delegate* – the efficiency advantage the matrix highlights (no round-trip to the RS to mint downstream authority).
- **Chain verification: offline-capable.** The verifier checks the supplied chain locally with no network dereferencing (subject to DID method and revocation-freshness caveats noted under #4).
- **Invocation: depends on the RS.** Invoking acts on the resource, which lives at the RS. So, if the RS is offline-capable (on a LAN etc) then the invocation can be offline.
- **Revocation: depends on the RS** (RS state change, per #6).

So zCaps are offline-capable precisely in the part (delegation) where the matrix says it matters most for efficiency, and online in the parts (invoke/revoke) that touch the protected resource.

Summary

zCaps are a mature, deployed realization of the object-capability model and they focus on the *mechanics* of delegated authority: chaining, monotonic attenuation, locally and cross-organizationally verifiable proof chains, strong proof-of-possession, and offline delegation.

zCaps are not focused on anything that is about *identity semantics* or *expressive policy* rather than raw authority flow: operator-vs-agent distinction is implicit (needs a companion VC layer), policy expression beyond simple action/time/URL attenuation is unstandardized and largely unused in practice (no native “undelegatable”, re-delegation rules, or advisory “please” semantics). For delegated authorization *for AI agents*, zCaps provide a good backbone for the authority graph, but would need (a) a richer, standardized caveat/policy vocabulary and (b) a companion identity-binding mechanism to fully satisfy the accountability and policy rows of the matrix.

input: UCAN

Chapter: UCAN

(Delegable Certificate Capabilities)

Overview

User-Controlled Authorization Network (UCAN) is a secure, local-first, user-originated, distributed authorization scheme that provides public-key-verifiable, delegable, expressive, and extensible capabilities in a certificate-capability style. A UCAN is a signed token (encoded as a DAG-CBOR payload inside a Varsig envelope, addressed by a CID) that associates a subject (sub) with a command (cmd) and policy (time bounds, arguments, etc.), and proves authority via a chain of delegations that ultimately roots in a resource owner.

UCAN adopts a certificate-capability model related to SPKI: there is no central authorization server, the resource owner is the root of trust for all their resources and certified capabilities, and delegation is self-certifying and offline-verifiable. The lifecycle is explicitly structured into four specs: **Delegation** (attest authority), **Invocation** (exercise authority), **Promise** (await and track results within invocations), and **Revocation** (break malicious or unwanted delegation chains).

Scorecard (UCAN vs. the zCaps template)

This table mirrors your zCaps "Scorecard" structure, but is based only on the UCAN specification set (overview + delegation + invocation + revocation).

Requirement	Verdict (UCAN)	Notes (from UCAN spec only)
Accountable (agent vs. principal/operator)	Partial	Every delegation edge is signed by the issuer DID and is part of a verifiable proof chain, but the data model does not distinguish "agent" from "operator"; both are generic principals (DIDs / Subjects).

		Authority provenance is explicit; legal semantics are external.
Resistant to confused deputy		
Represent authorization policies (“undelegatable”, re-delegation, advisory “please”)	Partial	UCAN defines “capability = subject × command × policy” and allows arbitrary policy expressions in the capability and arguments, but it does not standardize a caveat language for “no further delegation”, re-delegation rules, or advisory non-enforced guidance; those are left to resource-specific semantics.
Chainable	Yes	Delegation spec defines proof chains: UCANs act as self-certifying certificates where each link attests a subset of its delegator’s authority, forming a hash-linked chain rooted in a resource owner.

Cross-organizational / locally verifiable	Yes	UCAN uses DIDs as principals, content-addressed payloads (CIDs), and signatures over canonical DAG-CBOR; verification is local and does not require contacting a central authorization server, as long as proofs are resolvable (these can be “synced” easily by policy because usually implemented as a content-addressed k/v stores and/or NoDB).
Attenuated	Yes	The model requires that each delegation restate or attenuate capabilities; time bounds (nbf/exp) and the structure of the command/arguments provide axes for attenuation, and the spec explicitly defines attenuation as the process of constraining capabilities along the chain.
Self-revocable (holder or anyone in chain)	Yes (per Revocation spec)	Revocation is a first-class sub-spec: a delegable and revocable Revoker capability exists; when invoked, revocation “undoes a delegation, breaking a delegation chain for malicious users” by invalidating links in the chain rooted in that issuer. Who can revoke and how proofs of revocation propagate is defined at protocol level.

Composable (combine sources in one request)	Partial	A capability is defined as subject × command × policy, and the "authority" of a principal is the union of capabilities; the spec allows set-style merging of capability authorities and emphasizes compositionality across resources, but does not fully standardize multi-root combination semantics in a single invocation.
Authentication / Proof of Possession	Yes	Each UCAN must be signed by the issuer DID's key; invocation includes the UCAN and arguments, and executors validate signatures and chains. Replay prevention relies on nonces and time bounds; UIs/transport specs can also require binding invocation to a fresh challenge.
Privacy of delegation chain	No	Delegation chains are explicit proofs; the spec treats them as a provenance log, not a selectively-disclosable structure. Executors are expected to see the chain they validate; no selective-disclosure, encryption, or blinding mechanism is defined.

Offline-capable	Yes (for delegation/verification)	UCAN is explicitly “local-first” and partition-tolerant: delegations are self-contained certificates, verification is cryptographic and content-addressed; only dereferencing CIDs and applying revocation requires communication.

---

1. Accountable – agent vs. principal/operator

UCAN names all principals using DIDs and defines explicit roles: Issuer (iss), Audience (aud), Subject (sub), Owner, Executor, Invoker, Revoker, Validator, etc. Every UCAN must be signed with the secret key associated with the issuer’s DID; the delegation and revocation specs treat the chain of UCANs and revocation events as a provenance log of “who delegated what to whom” and “who revoked what,” with the caveat that those “who”s may be known only by a public key, or even one “published” only very locally. Accountability (to responsible persons) is conditional on the choice of identifiers/key-indirections, i.e. how much other information a given accountability-seeker has on each of those public keys.

However, the spec intentionally does not distinguish in the data model between “autonomous agent process” and “legal entity/operator”: a principal is simply a DID-identified agent, and an Owner is “a Subject that controls some external resource.” Binding those DIDs to human or organizational identities is out of scope for UCAN itself and would be accomplished via DID methods and higher-level identity systems, much like zCaps relies on DIDs/VCs for that distinction.

---

2. Ability to represent authorization policies

UCAN’s core definition of a capability is:

A capability is the association of an ability to a subject: subject × command × policy.

The Subject (sub) and Command (cmd) are required, and policy is expressed implicitly via arguments (args), command structure, and possibly resource-specific semantics; the examples show policies such as “from this email address” and “to @example.com” encoded as a JSON logic expression in the policy field.

The spec explicitly reserves `/ucan` as a command namespace for “community-blessed commands” such as revocation, but it does not standardize a general caveat vocabulary that, for example, would express “no further delegation” or “depth limit” inside the token itself. Delegation is constrained by attenuation rules (each step must restate or constrain capabilities), and the Revocation spec defines how delegators can later invalidate delegations, but advisory “please don’t re-delegate” semantics or non-enforced guidance are not part of the normative model; they would live in out-of-band governance or higher-level conventions.

---

3. Chainable

UCAN is explicitly built around a delegation chain model: the Delegation spec (linked from the main page) defines how authority is passed in a partition-tolerant way, and the main spec describes “proofs” as cryptographically verifiable chains that show either direct ownership of a resource or delegation from an owner. Root capability issuers function as distributed roots of trust, and the chain is “by definition a provenance log.”

Attenuation is defined as the requirement that each direct delegation restates or diminishes capabilities; combined with signatures and references (via content IDs or embedded proofs), this yields a multi-hop delegation chain: owner → agent → sub-agent → etc., all verifiable offline, with the Subject (sub) field marking whose authority the chain is about.

---

4. Cross-organizational / locally verifiable

UCAN is designed for distributed, local-first systems: it explicitly positions itself as trustless, secure, and partition-tolerant, with no Authorization Server acting as an intermediary. In the “Inversion of Control” section, the spec emphasizes that the resource owner (resource server) grants authority directly to agents; UCAN requests are “self-contained” and do not require an intermediary authorization check.

Principals are DIDs, and UCAN tokens are content-addressed with CIDv1, SHA-256, base58btc, and DAG-CBOR encoding; verification consists of canonical decoding, signature checking against the DID’s key, and walking the delegation chain using CIDs

(local store, DHT, gossip, REST, etc.). None of this assumes a shared IAM system: cross-organization delegation is just DID-to-DID certificates, and verification is local as long as proofs are resolvable.

---

5. Attenuated

UCAN defines **attenuation** as "the process of constraining the capabilities in a delegation chain"; each direct delegation must either "directly restate or attenuate (diminish) its capabilities." Concretely:

- Capabilities are subject × command × policy; a child delegation must not widen that set relative to its parent.
- Time is bounded by nbf (not-before) and exp (expires); the "validity interval" is the intersection across the chain: latest nbf to earliest exp.
- Commands have a segment structure where shorter commands "prove" longer paths in the same namespace (e.g., `/crud` can prove `/crud/read` but not `/stack/pop`), so attenuation in command space is monotonic.

The "Authority" section stipulates that merging capability authorities follows set semantics: the union of capability sets, and "every unique delegated capability MUST have equal or narrower capabilities from their delegator." That is, UCAN's expressiveness is encoded in command names, arguments, and resource URIs, but the core invariant is monotonic attenuation along the chain.

---

6. Self-revocable

Revocation is a first-class part of the lifecycle: the Revocation spec is "RECOMMENDED" and described as "Undo a delegation, breaking a delegation chain for malicious users." The main spec introduces a **Revoker** role as "The Issuer listed in a proof chain that revokes a UCAN."

In the lifecycle example, a Delegation Agent first delegates to Alice, who delegates to Bob; later, Alice invokes a revoke operation with proofs to the Delegation Agent, after which Bob's subsequent invocations are rejected. This illustrates that a party in the chain (here, Alice) can trigger revocation of a downstream delegation, and executors must validate revocation at execution time. The precise mechanics (how revocation notices are represented, how they are discovered) are defined in the Revocation sub-spec, but architecturally UCAN supports revocation by issuers in the chain, with revocation treated as breaking links so later invocations fail validation.

---

7. Composable

UCAN's notion of "authority" is explicitly set-theoretic: "The set of capabilities delegated by a UCAN is called its 'authority'," and "Merging capability authorities MUST follow set semantics, where the result includes all capabilities from the input authorities." Capabilities can overlap, and the effective authority is their union; "Capability authorities can be combined in any order, with the result always being at least as broad as each of the original authorities."

At the same time, each individual UCAN payload has a single command (cmd) and argument set (args), and the spec is focused on single-command capabilities. The lifecycle and invocation specs define how an invocation exercises authority for a specific command and subject, not a general multi-root, multi-chain composition model. Compositionality is primarily about being able to treat resources consistently whether they are co-located or not, and about merging authorities in an abstract set sense; combining distinct roots in one invocation is possible in principle but not deeply specified as a separate pattern.

Nice-to-haves

Authentication / Proof of Possession

The UCAN envelope requires:

- A Varsig header describing the signature scheme.
- A TokenPayload with iss, aud, sub, cmd, args, nonce, meta, nbf, exp.

Every UCAN must be signed by the issuer's key, and executors validate this signature over the canonical DAG-CBOR representation; combined with DID-based identification, this ensures that only someone with the issuer's secret key could have produced the UCAN.

Replay protection is handled at least at three levels in the core spec:

- **Nonce**: a required, random or monotonic field that ensures unique CIDs and supports replay detection.
- **Time bounds**: nbf and exp define a validity interval; tokens outside that window must be treated as invalid.

- **Invocation layer**: the spec's Q&A explicitly asks "What prevents an unauthorized party from using an intercepted UCAN?" and "What prevents replay attacks?" and answers in terms of signed invocations and nonces.

Together, these give UCAN a proof-of-possession model comparable in intent to zCaps' HTTP signatures / DI proofs, but expressed in the UCAN envelope and transport.

Privacy of the delegation chain

The spec is explicit that delegation chains are provenance logs and that UCANs "do not offer confinement," and that delegates "can further sub-delegate their authority to others without alerting their delegator." Executors must validate the chain at execution time, including ownership of external resources, and the ability to revoke downstream UCANs is a "last resort."

There is no mechanism described for selectively hiding or redacting parts of the chain from executors; validation assumes access to the relevant proofs (directly or via CID resolution), and the chain is considered positive evidence ("proofs are positive evidence… cryptographically verifiable chains"). So, like zCaps, UCAN treats the chain as fully visible to the validating party (although additional information, if any, associated with the actors in the chain are usually out-of-bound as non-key DID document contents, verifiable credentials, or other identifier-anchored metadata).

Offline-capable

UCAN is explicitly framed as "local-first" and designed for partition tolerance and replicated state: one of the listed advantages of inversion of control is "Partition tolerance, support for replicated data and machines."

- **Delegation**: UCANs are self-certifying certificates; they can be created and passed around offline as long as keys and DIDs are known.
- **Invocation/validation**: validation is local over signed, content-addressed tokens; if proofs are locally available (e.g., via a local store or cached CIDs), no online authority is needed.
- **Revocation**: requires propagating revocation information; the spec notes that "the ability to revoke some or all downstream UCANs exists as a last resort," but does not require an always-online central point. Implementations are free to use DHTs, gossip, etc., for revocation dissemination.

In short, within the confines of the spec itself, UCAN provides offline delegation and local validation, with revocation and resource access depending on whatever communication pattern the application adopts—very much in line with the

certificate-capability model described in the Motivation and Inversion-of-Control sections.

Spec-level feature differences: UCAN vs. zCAP-LD

This section highlights **features that exist in one spec but not the other**, rather than general similarities.

Multiple actions per capability

- **zCAP-LD:**
  - A delegated zcap MAY include an `allowedAction` property that is either a **single string or an array of strings** "that each express an action" the controller may perform when invoking the capability.
  - Example from the spec: `"allowedAction": ["write", "read"]`.
  - Verifiers MUST ensure that `allowedAction` in a delegated zcap is not less restrictive than in its parent capability.
  - **Net:** a single delegated zcap can directly encode multiple discrete actions, e.g., "read and write but not delete."
- **UCAN:**
  - The token payload has a single `cmd` field: a required `String`, "The Command to eventually invoke."
  - A capability is `subject × command × policy`; the spec shows multiple distinct commands (`/msg/send`, `/crud/read`, `/crud/mutate`, etc.), but **each UCAN has exactly one `cmd` string.**
  - The spec does not define any way for `cmd` to be a list/set of commands. To grant multiple methods, UCAN relies on either:
    - A more general command (e.g. `/crud`) whose **policy/arguments** restrict which operations are allowed, or
    - Multiple UCANs, each with its own `cmd`.

Asymmetry: zCAP-LD has built-in multi-action support via `allowedAction` arrays; UCAN enforces a single `cmd` per token and pushes multi-method expressiveness into policy or multiple tokens.

Non-Overlapping Features

Resource scoping: URL attenuation vs command lattice

- **zCAP-LD:**
  - Each delegated zcap has an `invocationTarget` URI.
  - For attenuation, a delegated capability's `invocationTarget` MUST either equal the parent's or be a **prefix extension**: the parent's URI as prefix plus an added path or query suffix.
  - The spec spells out "URL Path- or Query-based Attenuation": suffix starts with `/` or `?` (or `&` if queries exist), and several examples show hierarchical narrowing like `/bars/123` → `/bars/123/bazzes/456?day=tuesday&hour=12`.
  - **Net:** URL path/query structure is directly part of the attenuation rule in the data model.
- **UCAN:**
  - UCAN does not define an `invocationTarget` field on the capability equivalent to zCAP; instead it models capabilities via `cmd` plus arguments, and treats external resources as being governed by an Owner/Subject relationship and command semantics.
  - It defines a **command lattice**: commands are path-like strings (e.g. `/crud`, `/crud/read`, `/crud/mutate/update`), and "segment structure is important since shorter commands prove longer paths" while not proving unrelated namespaces (e.g. `/crud` must not prove `/stack/pop`).
  - Attenuation is defined abstractly: each direct delegation MUST restate or attenuate capabilities, and authorities are sets that can be intersected/unioned under those constraints; resource scoping is left to how `cmd` and arguments are interpreted, not to a fixed URL suffix rule.

Asymmetry: zCAP-LD hard-codes URL path/query attenuation on `invocationTarget`; UCAN does not, instead using a command and authority lattice without normative URL rules in the core spec.

Capability composition and "Powerline"-style authority union

- **UCAN:**
  - The spec defines a UCAN's "authority" as "the set of capabilities delegated by a UCAN."

    - It states that **merging capability authorities MUST follow set semantics**, where the result includes all capabilities from the input authorities; capability sets can overlap, and the merged authority is at least as broad as each individual authority, subject to attenuation rules.
    - This is explicitly used to support **compositionality across resources**, with no distinction between co-located and distributed resources in the authority model; the same capability description applies regardless of location.
- **zCAP-LD:**
    - The delegated-capability section defines a single `parentCapability` per delegated zcap and a single `invocationTarget` per capability.
    - The capability delegation proof's `capabilityChain` is specified as "THIS delegated zcap's parent and its parent and so on up to the root zcap"; there is no defined algebra for merging multiple chains or capabilities into a new combined authority.

Asymmetry: UCAN has a spec-level notion of capability authorities as sets with explicit union semantics ("merge authorities"), supporting multi-UCAN compositionality; zCAP-LD defines only single-root, single-target chains, and does not specify a way to combine multiple capability authorities into a single, composed capability at the spec level.

Container and signature format

- **UCAN:**
    - UCANs are encoded as **DAG-CBOR** and identified by **CIDv1** (SHA-256 + `base58btc`); they are native IPLD/IPFS objects.
    - A UCAN token is a Varsig envelope containing:
        - `.0`: signature bytes,
        - `.1.h`: Varsig header (algorithm, encoding, key info),
        - `.1.ucan/@`: `TokenPayload` with `iss`, `aud`, `sub`, `cmd`, `args`, `nonce`, `meta`, `nbf`, `exp`.
    - The spec lists a cryptosuite (hash and signature algorithms, DID method) and references a general Varsig specification for signature semantics.
- **zCAP-LD:**
    - Capabilities are **JSON-LD documents** with `@context` including `https://w3id.org/zcap/v1` and an appropriate security context (e.g. Ed25519 2020).

- Delegated capabilities are signed using **Data Integrity proofs** (`capabilityDelegation` proof), with fields such as `type`, `created`, `verificationMethod`, `proofPurpose`, `proofValue`, and `capabilityChain`.
- There is no CID/IPLD container requirement in the delegated-capability section.

Asymmetry: UCAN standardizes a content-addressed DAG-CBOR + Varsig envelope format; zCAP-LD standardizes JSON-LD + Data Integrity proofs and does not define a CID/IPLD wrapper in the delegated-capability spec.

Revocation as command vs revocation endpoint

- **UCAN:**
  - Revocation is a named part of the lifecycle: `[RECOMMENDED] Revocation – Undo a delegation, breaking a delegation chain for malicious users." [page:0]
  - There is a **Revoker** role ("The Issuer listed in a proof chain that revokes a UCAN") and a reserved `/ucan` command namespace for community-blessed commands, including revocation (e.g. `/ucan/revoke`).
  - Revocation is expressed as UCAN operations interpreted by executors, i.e., revocation is a protocol-level command in the same capability model.
- **zCAP-LD:**
  - Delegated-capability spec uses expiration (`expires`) as the built-in lifetime limit; verifiers MUST ensure a delegated capability has not expired and that `expires` is no less restrictive than the parent.
  - For revocation, the spec defines an **RS-local endpoint pattern**:
    - The root `invocationTarget` SHOULD have a `/zcaps/revocations` subpath.
    - Any controller in the chain may revoke by POSTing the zcap to that endpoint.
  - Revocation is not modeled as a capability command in the zCAP-LD data model; it is a server policy and state change.

Asymmetry: UCAN treats revocation as a first-class command (`/ucan/revoke`) and role (Revoker) in the same capability model; zCAP-LD treats revocation as an RS-specific endpoint convention plus standard expiration, without a revocation command in the capability space.

Chain structure and storage

- **zCAP-LD:**
  - Each delegated zcap has a `parentCapability` pointing to either the root or another delegated zcap (by ID).
  - The `capabilityDelegation` proof contains a `capabilityChain` array that MUST:
    - Include the **root zcap ID**.
    - Include every ancestor of the delegated zcap by ID, except the immediate parent, which must be **fully embedded** in the delegated zcap.
  - Verifiers MUST NOT be required to perform network requests or DB lookups to dereference capability IDs during verification; the full chain must be present in the invocation.
- **UCAN:**
  - UCAN uses a **Delegation Store** addressed by CIDs; UCANs can be resolved from local stores, DHTs, gossip, or REST endpoints.
  - The chain is logically a sequence of proofs (UCANs) from an Owner to the Subject, but the spec does not force a particular embedding pattern; proofs may be carried inline or referenced by CID.

Asymmetry: zCAP-LD fixes the representation of the chain (parent embedded, others by ID, no network access required at verification time); UCAN defines chains logically and relies on content-addressed storage and transport to provide proofs, without prescribing embedding vs reference structure in the same way.

Invocation mechanism

- **zCAP-LD:**
  - Defines invocation both for HTTP and Data Integrity:
    - HTTP: a `capability-invocation` header with `id` (root or delegated zcap ID) and `action`, plus the HTTP request URL as `invocationTarget`. Delegated zcaps are attached via header parameter (e.g., `capability="zcap:…"` containing the compressed delegated zcap).
    - Data Integrity: a `capabilityInvocation` proof on arbitrary JSON-LD, with `capability` (ID or embedded zcap), `invocationTarget`, and `capabilityAction`.
  - Invocation semantics are expressed via Data Integrity or HTTP signatures on top of standard web protocols.

- **UCAN:**
    - Uses the same UCAN envelope and payload for both Delegation and Invocation; the Invocation spec describes how an Invoker sends UCAN(s) plus `args` to an Executor, which validates signatures, chains, nonce, time bounds, and revocation.
    - Transport is left open (e.g., RPC, HTTP, message queues), but there is no use of Data Integrity proofs; the UCAN token itself is the signed authorization object.

Asymmetry: zCAP-LD defines invocation via Data Integrity and HTTP signature conventions with explicit `capabilityInvocation` semantics; UCAN defines invocation by reusing its own signed UCAN envelope and a dedicated Invocation spec, and does not use Data Integrity in the core model.

Powerline

In UCAN, "Powerline" is a delegation pattern where **one principal designates another principal as the automatic recipient of all (or a class of) future delegations**, so that whenever any issuer delegates into that "line," authority transparently flows to the designated agent without them needing to be explicitly named in each delegation.

More concretely, based on the UCAN delegation work:

- A **Powerline** is defined as "a pattern for automatically delegating all future delegations to another agent regardless of Subject."
- It is implemented by **explicitly setting certain fields in the delegation** (in the UCAN delegation spec) so that:
    - The Powerline's "owner" or "sink" agent is fixed.
    - Any UCAN that delegates along that line effectively grants authority that can be exercised by, or further delegated by, that sink agent, even though individual delegations may have different Subjects.

Intuitively:

- Think of a Powerline as a **standing, path-like delegation conduit**: once established, it says "whoever gets delegated along this line, route the authority to X."
- This allows building systems where, say, a storage service or orchestration agent automatically receives all capabilities granted to users in a particular namespace,

without every delegator having to explicitly name that agent as a co-delegate each time.

The key points that make it a “pattern” rather than a new primitive:

- It’s **built from normal UCAN pieces** (principals, subjects, delegation rules); you implement it by choosing a convention for which fields identify the Powerline and how executors interpret those delegations.
- It **changes how authority propagates**: instead of only flowing “along the Subject,” it also flows along this designated “line,” which can be used to centralize audit, enforcement, or mediation logic in a specific agent.

So, in contrast to simple one-off delegation, a UCAN Powerline is about **pre-wiring a reusable delegation path to a particular agent**, so that future delegations that match the pattern automatically extend that agent’s effective authority, without requiring per-delegation customization.

Asymmetry: zCAP-LD has no analog to the Powerline pattern.

Appendix

Earlier I said:

I used the following prompt to have perplexity.ai do my homework. I have reviewed the text and made a few small edits, such as Oxford comma, reformatting the table, and secret key instead of private key. (Note that I skipped the section on Promises because it’s marked as a draft.)

Using the uploaded document as a template, produce a document that contrast the contents of https://ucan.xyz/specification/ including the material at the links for delegation, invocation, and revocation.

That prompt resulted in some significant deviations from the UCAN spec because the AI used additional, out of date sources. I revised the prompt to add, “Repeat the analysis using only https://ucan.xyz/specification/ for information on UCAN.”

When you produced the document based on the prompt, "Using the uploaded document as a template, produce a document that contrast the contents of https://ucan.xyz/specification/ including the material at the links for delegation, invocation, and revocation." you used additional sources of information for UCAN, many of which are outdated.

Repeat the analysis using only https://ucan.xyz/specification/ for information on UCAN.

I also asked perplexity.ai

In your zcap/UCAN comparison you didn't describe features that are in one but not the other, such as multiple allowed actions in zcap vs. only a single command string in UCAN or the Powerline feature in UCAN. Create such a document using only the given spec for UCAN and https://w3c-ccg.github.io/zcap-spec/#delegated-capability for zcap.

The answer is in the Non-Overlapping Features section.

Alan Karp

input: Cedar policy language

Chapter: The Cedar policy language

Overview

Cedar is an open-source authorization policy language and evaluation engine created by IAM specialists at Amazon with formal methods training (used in AWS Verified Permissions; the language syntax itself is now a CNCF candidate project). A Cedar policy answers "**may** this principal **perform** this action **on** this resource **under** these conditions (context)" using `permit` and `forbid` statements validated against a typed schema. Evaluation is deterministic, default-deny, and `forbid` always overrides `permit`. Over time, the core Cedar language and its canonical reference implementation in Rust have been generalized from Amazon's production implementation in Keycloak/AWS contexts to be hardened and governed at CNCF within the Linux Foundation (as mentioned above); a cloud-agnostic IAM framework generalizing the Keycloak/AWS tooling is governed in a dedicated Linux Foundation Project called Janssen, which also governs and develops the younger "Cedarling," a lightweight version of Cedar's enforcement engine that can be run "practically anywhere" as a WASM component inside of not just servers but even clients and agents/harnesses.

//TODO: add section explaining that machine-readable policy evaluation CAN be construed to include identity-blind/unauthenticated capabilities, OR AT LEAST to help such tooling be intelligible to agents and unfamiliar callers in a cross-org use-case.

Cedar has no native, enforced delegation primitive. It is a policy expression language plus a Policy Decision Point (PDP), with no native concept of a credential, a holder, a signature, or a delegation chain. It is *identity-based* in the classic sense: the verifier decides access by evaluating who the principal is (and their attributes and group memberships, as made known to the verifier by tokens from server-side tooling like Janssen) against policies the verifier holds, in contrast to possession-based models like zCaps, where presenting a valid signed capability constitutes the "just-in-time" proof of authority. This identity-vs-possession distinction drives nearly every row of the scorecard.

Cedar is therefore evaluated here **as a component in a delegation stack**: Cedar supplies policy expression and local evaluation in cross-organization use cases where policy must be made intelligible to callers and counterparties across policy boundaries; the delegation mechanics (chain construction, portable proof, holder-side revocation) must be supplied by a carrier layer around it, but the policy anchoring the semantics of those delegations and error messages can be more meaningfully communicated by the combination of the layers.

Three reference points are used:

- the Cedar language and documentation (the "spec": syntax, schema, validation, templates, evaluation semantics),

- the Janssen Project Cedarling, the most relevant deployed embedding: a lightweight, embeddable PDP (Rust core with WASM, mobile, Python and Java bindings) that implements Token-Based Access Control (TBAC), deriving Cedar principal entities from JWTs issued by trusted issuers and evaluating policies locally at the edge, and
- Clawdrey Hepburn, a first-person agentic case study: an autonomous AI agent whose operator-defined action boundaries (which accounts it may follow, what it may do with its own credentials and payment card) are enforced by an embedded Cedar engine checked before every action, with a typed schema modeling the agent, its operator, and its action space.

Where Cedar (the language) and runtime-evaluation deployments like Cedarling differ in what they provide, this is called out, since several matrix rows are satisfiable only by the deployment layer.

Scorecard

#	Requirement	Verdict	Notes
1	**Accountable** (agent vs. principal/operator)	Yes (at the modeling level)	The schema can type the distinction directly (e.g. `entity Agent in [User]`); Cedarling derives separate User and Workload principals from the id_token and access_token and can require both to be authorized. But the distinction is data the verifier trusts an Authorization Server and token exchange to provide, not a cryptographically attributable delegation edge.
	Resistant to confused deputy		
2	**Represent authorization policies** (embed/link; "undelegatable", re-delegation rules, advisory "please")	Yes, except advisory	This is Cedar's core competence. "Do not delegate further" and re-delegation rules are expressible if delegation itself is modeled as an action. No advisory tier: decisions are binary permit/forbid, with no obligations or native "honor-if-possible" semantics.
3	**Chainable**	No (stack-dependent)	No delegation primitive, no chain artifact. Delegation can be modeled as verifier-side **state** (entities or instantiated policy templates), but chains are not portable, signed, or first-class. In a delegation stack, the chain must live in the carrier

#	Requirement	Verdict	Notes
			or elsewhere in the Authorization stack; Cedar evaluates each hop atomically.
4	**Cross-organizational / locally verifiable**	Partial	Evaluation is fully local (Cedarling runs in browsers, mobile, gateways; multi-issuer JWTs supported without shared IAM accounts). But the policy store is unilateral, that is the verifier's: there is no standard or guidance for exchanging policy semantics across organizations in an adhoc way, and trusted issuers must be pre-configured, BUT core contributors do contribute to and recommend using AuthZEN for federation and runtime issuer-list
5	**Attenuated**	Partial	Forbid-overrides-permit gives monotonic restriction: layering additional forbid policies (or a stricter verifier-side policy set) can only shrink authority, never expand it. But Cedar has no runtime check that a delegatee's grant is a subset of the delegator's; subset/equivalence can be proven offline with Cedar's symbolic analysis tooling. Attenuation enforcement is a stack responsibility.
6	**Self-revocable** (holder or somebody in chain)	Partial	Revocation is a policy store change (remove a permit or add a forbid), immediate at the next evaluation and very fine-grained. But it is performed by whoever administers the policy store, not natively by a delegator or delegatee in the chain; mapping chain participants to revocation rights is a stack design task. Cascade to downstream grants is not native either: it holds only if delegation is modeled as linked entities whose policy re-checks the ancestry.
7	**Composable** (combine sources in one request)	Yes	A request is evaluated against the union of all loaded policies; permits can come from independent sources; principals can be in many groups/roles simultaneously. Cedarling's `authorize_multi_issuer` evaluates JWTs from multiple issuers in a single request. No single-role restriction.

#	Requirement	Verdict	Notes
+	Authentication / proof of possession	No (out of scope)	Cedar does not authenticate. Cedarling validates JWT signatures (proof of issuance by a trusted issuer), which is not holder proof of possession; PoP binding is left to the token layer.
+	Privacy of delegation chain	Possible (by architecture)	There is no chain in the data model, so nothing is carried in the request: a delegatee presents only their own token/identity. Intermediate delegatees need not learn the rest of the chain. The flip side: the verifier's policy store and entity data must hold whatever graph exists, so the verifier sees everything.
+	Offline-capable	Partial	Evaluation is fully offline once the policy store is loaded (this is Cedarling's headline feature). But *granting* a delegation means writing to a policy store, which is an online, administrative act, the opposite of zCaps' offline delegation-by-signing.

(Cedar) detailed evaluation

1. Accountable, agent vs. principal/operator

Differentiation is a schema design choice, made trivially. Cedar entities are typed and hierarchical, so the agent/operator relationship can be modeled directly, for example `entity Agent in [User] { model: String, runtime: String, trust_level: String }`, which is exactly the pattern the Clawdrey case study uses: the agent is a typed principal nested under its human operator, and policies can condition on either or both. Cedarling operationalizes the same split out of the box: it constructs a **User** principal from the OIDC id_token and a separate **Workload** principal from the OAuth access token (the software acting on behalf of the person, or autonomously), and its default decision logic can require that *both* the person and the workload be authorized for the request to proceed. That is precisely the agent-vs-operator distinction the matrix asks for, expressed natively in the data model rather than by convention.

What Cedar does not provide is cryptographic accountability for the *delegation* itself. There is no signed delegation edge: the "fact" that operator O stands behind agent A is an entity attribute or a token claim that the verifier chose to trust. Attribution quality is therefore exactly the attribution

quality of the surrounding identity layer (the JWT issuer, in Cedarling's case). An auditor can read the decision logs (Cedarling logs every decision with diagnostics) and reconstruct *which policy permitted which principal*, but cannot derive a non-repudiable "this delegator authorized this delegatee" proof from the Cedar artifacts alone. The accountable party for any grant is, structurally, the policy store administrator.

2. Ability to represent authorization policies

Policies are first-class, schema-validated, and arbitrarily conditional on principal, action, resource and request context: attribute comparisons, group membership (`in`), set operations, string and IP functions, and explicit `forbid` statements that override any permit. Policies can be embedded with the verifier or distributed to it (Cedarling fetches signed policy stores over HTTPS or from a Lock Server), satisfying the embed-or-link condition.

Against the matrix's specific examples:

- **"Do not delegate further" / "this permission is undelegatable":** expressible, with one precondition: delegation must be modeled as an action in the schema (e.g. `action delegate appliesTo { principal: [Agent], resource: [Permission] }`). Then `forbid (principal, action == Action::"delegate", resource == Permission::"x");` makes permission x undelegatable, and since forbid overrides permit, no later grant can reopen it. Depth limits are expressible the same way if depth is carried as an attribute. The precondition matters: Cedar can only forbid delegations that the stack routes through a Cedar check.
- **Re-delegation policies:** same mechanism, with the full expressiveness of the language (delegate only to principals in a given group, only with attenuated scopes, only before an expiry, and so on).
- **Advisory "please" rules (a request carrying desired re-delegation rules whose violation is not refused):** not supported. Cedar evaluation returns a binary Allow/Deny plus diagnostics; there is no obligations or advice tier in the decision (a notable contrast with XACML), and policy annotations like `@id(...)` are non-semantic metadata. An advisory tier would have to be built beside Cedar, not in it.
- **"Who gets to know what is a governance topic":** Cedar is agnostic; it evaluates whatever entity data it is handed. Disclosure governance sits in the surrounding stack (which claims go into tokens, which attributes into the policy store).

One genuinely distinctive property deserves mention here: Cedar is **analyzable**. The language was deliberately kept non-Turing-complete (no loops, no recursion, guaranteed termination) and its evaluator is formally verified, so policy sets can be subjected to automated reasoning: does policy set A permit anything B does not, are these two sets equivalent, is this forbid ever reachable. For a delegation context, that means delegation rules can be *audited and proven* offline, which no other technology in this survey offers.

3. Chainable

Evaluated as-is: no. Cedar has no delegation primitive. There is no parent pointer, no chain, no signed artifact that one party hands to another. Two in-language mechanisms get part of the way and are worth naming precisely:

- **Policy templates.** A template with `?principal` / `?resource` slots can be instantiated at runtime to grant a specific principal access to a specific resource. Template instantiation *is* a grant operation and can be wired up as the delegation act ("delegator triggers instantiation of a template scoped to the delegatee"). But instantiation is an administrative write to the policy store, performed by whoever holds write access, and the resulting policy records no ancestry.
- **Delegation modeled as data.** The schema can define delegation entities (delegator, delegatee, scope, expiry, parent reference) and policies that grant access when a valid delegation entity exists. This can represent chains of arbitrary depth, including agent-to-subagent links. But the chain is then application state in the verifier's entity store: not portable, not signed, not independently verifiable, and entirely a bespoke convention.

So in a delegation stack, the chain must live in the carrier layer (signed tokens or credentials linked hop to hop), and Cedar's role is to evaluate the rules *at* each hop: may this delegator delegate this, may this delegatee exercise it. Cedar is the rulebook for the chain, not the chain.

4. Cross-organizational / locally verifiable

Half of this requirement is Cedarling's headline feature; the other half is structurally missing.

Locally verifiable: yes. Cedar evaluation requires no network call: policies, schema and entity data are in memory, and decisions are sub-millisecond (for simple policies only). Cedarling pushes this to the edge deliberately: a WASM/mobile/server-embeddable PDP that loads a policy store at startup and answers authorization questions locally, so an API gateway, browser or device enforces policy without phoning a central authorization server per request. JWT signature validation against trusted issuers is also local (cached JWKS).

Cross-organizational: partial. Cedarling's TBAC model is genuinely multi-party at the *evidence* level: `authorize_multi_issuer` accepts JWTs from several different trusted issuers in one request and maps their claims to Cedar entities, so organization A's verifier can consume identity evidence issued by organization B without B having an account in A's IAM. But the *policy* level is unilateral: the policy store (schema, policies, trusted issuer list) belongs to the verifier alone. There is no standard for one organization to hand another a policy fragment with agreed semantics, no shared schema federation, and the trusted issuer list is pre-configuration, not dynamic trust establishment. Two organizations can interoperate, but only after bilateral setup; the matrix's "express authorization policies without accounts on each other's IAM systems" is met for authentication evidence and unmet for policy exchange.

5. Attenuated

Cedar has one structural property that maps beautifully onto attenuation, and one missing property that prevents a full Yes.

The structural property is **monotonic restriction by composition**: `forbid` overrides `permit`, and evaluation is over the union of all policies. Adding policies to a set can therefore only ever *shrink* what a forbid touches and a verifier (or any later layer) can stack stricter policy on top of a base grant with a guarantee that the stack cannot widen authority. This is a natural fit for the delegation-stack pattern in which each hop attaches further restrictions: express the delegator's grant as the base policy set and each subsequent restriction as additional forbids or narrower permits evaluated together.

What is missing is **enforced subset semantics between grants**. If a delegation is represented as "delegatee gets policy set B, derived from delegator's policy set A", nothing in the Cedar runtime checks $B \subseteq A$; an over-broad B is just another valid policy set. The check is possible, and this is where Cedar is unique: its analyzability means "B permits nothing A does not" is a decidable question answerable by symbolic analysis tooling at delegation time or audit time. But it is an offline/toolchain guarantee the stack must invoke, not a property the evaluator enforces per request the way a zCaps verifier enforces allowedAction narrowing.

Expressiveness of attenuation, once enforced, is far richer than URL-suffix narrowing: any condition the language can state (attribute ranges, context conditions, time windows, resource groups) can be the attenuation axis.

6. Self-revocable

Revocation in Cedar terms is straightforward and immediate: delete the permit (or the template instantiation, or the delegation entity) or add a forbid, and the very next evaluation reflects it. Forbid-overrides-permit makes "revoke" particularly clean: a targeted forbid kills a grant without having to find and remove every permit that contributed to it. Granularity is excellent: one delegatee, one action, one resource, one condition.

The matrix question, however, is *who* can revoke. Natively: whoever can write to the policy store. That is an administrative role, not a chain position. A delegator can revoke what they delegated only if the stack maps "delegator of grant X" to "may modify/forbid grant X in the store", which Cedar can itself express as policy (delegation management actions authorized by Cedar policies, the recursive pattern) but does not provide out of the box. Propagation is also a deployment property: Cedarling instances cache their policy store and pick up changes on refresh or via Lock Server push, so revocation latency equals policy distribution latency. Within a single trust domain this is near-immediate; across many edge PDPs it is eventually consistent.

On **cascade** (does revoking a grant revoke everything delegated below it): Cedar has no built-in notion of a chain, so there is no built-in cascade either. Whether revocation propagates downstream depends on how delegation is modeled. If each grant is an independent policy,

revoking Alice-to-Bob leaves Bob-to-Carol untouched, and Carol keeps her access until that grant is separately removed. Cascade is achievable by modeling delegation as data with parent links plus a permit policy that succeeds only when the whole ancestry is still valid: each delegation is an entity carrying a parent reference, and Carol's evaluation walks up the chain, so revoking Alice-to-Bob cascades to Bob-to-Carol because the walk hits the now-missing or now-forbidden link and fails closed. This is a contrast with zCaps, where cascade is structural (a delegated capability embeds its parent, so rejecting any link fails everything below it by construction); in Cedar cascade is a property you build into the policy and entity model, not one the engine provides.

Net: revocation is a strength of the centralized-policy model (no token to chase, no expiry to wait out, unlike credential-based systems where an issued artifact is "out there"), but holder/chain self-revocation is a stack construct, not a Cedar feature.

7. Composable

A clear yes. Cedar evaluates every request against the union of all loaded policies; an Allow needs at least one satisfied permit and no satisfied forbid, and the satisfied permit(s) can originate from any source loaded into the store. Principals can belong to arbitrarily many groups and roles simultaneously (the matrix's RBAC single-role counter-example does not apply: Cedar's `in` hierarchy is a DAG, not a single assignment). Cedarling extends composition to the evidence layer: `authorize_multi_issuer` lets one request carry tokens from multiple independent issuers, each mapped to entities, so authority deriving from several principals or organizations is combined in a single decision.

The one caveat mirrors row 4: all the composed sources must already be present in (or mapped into) the verifier's policy store and trusted issuer configuration by stable identifiers. Composition is native at decision time, but onboarding a new source of authority is configuration.

Nice-to-haves

Authentication / proof of possession

Out of scope for Cedar by design: the engine assumes the principal has already been authenticated and evaluates the request it is handed. Cedarling adds issuance-side verification: JWT signature validation against pre-configured trusted issuers, aud/sub cross-checks between tokens, optional status checks, and exp/nbf checks. This proves the tokens were issued by a trusted party and are current, not that the presenter is the rightful holder. Sender-constrained tokens (DPoP/mTLS-bound) can be layered in by the token infrastructure, but Cedar/Cedarling does not itself verify possession.

Privacy of the delegation chain

An interesting inversion of the credential-based technologies. Because nothing is carried in the request, the matrix's desired property (the intermediate delegatee does not learn the rest of the chain) falls out of the architecture almost for free: each delegatee authenticates as themselves and presents their own tokens; whatever delegation graph exists is reconstructed verifier-side from the policy store and entity data. The principal (as policy author) and the verifier can know the full story while the token carrier knows only their own grant.

The cost is the mirror image: the verifier's policy store must contain the graph, so the verifier (and the policy store infrastructure) sees everything, and chain privacy *from the verifier* is impossible. Whether this counts as satisfying the row depends on which party the subgroup is trying to blind; for the stated formulation (hide the chain from the intermediate agent, principal and verifier know all) Cedar-style architectures actually do well.

Offline-capable

- **Evaluation/use: fully offline.** This is Cedarling's reason for existing: the PDP is embedded (browser WASM, mobile, gateway, local process) and decides locally with no network round trip. Local-machine and LAN-only agentic use cases are well served; the Clawdrey deployment is exactly this shape (an agent on a single machine consulting its local Cedar engine before each action).
- **Delegation/granting: online and administrative.** Creating a new grant means writing to a policy store and distributing it. There is no offline delegate-by-signing; the efficiency advantage the matrix highlights (delegate without contacting anyone) is absent.
- **Revocation: online to the store, then eventually consistent** to edge PDPs on refresh.

Summary

Cedar is, almost row by row, the complement of the capability-based technologies in this survey. It is strongest exactly where zCaps are weakest: rich, analyzable, schema-validated policy expression (including "undelegatable" and re-delegation rules), native agent-vs-operator modeling, composition of authority from multiple sources in one decision, fine-grained immediate revocation, and fully local evaluation at the edge. It is weakest exactly where zCaps are strongest: there is no delegation chain, no portable signed grant, no holder-side revocation, no offline delegation, and no cryptographic attribution of who authorized whom; all of these are verifier-side state administered through a policy store.

For delegated authorization for AI agents, the implication is that Cedar is not a candidate *delegation data model* but a strong candidate *policy layer inside one*: a carrier technology supplies the chain, proof and portability, while Cedar supplies the rulebook each hop is checked against, with the unusual bonus that those rules can be formally analyzed (delegation attenuation provable offline). The Cedarling deployment shows the integration pattern already in production form (policies travel to the edge, identity evidence travels as multi-issuer JWTs,

decisions stay local), and the Clawdrey case study shows the agentic pattern (operator-authored Cedar policies as the hard boundary on an autonomous agent's action space). What no current Cedar deployment shows is the missing piece the matrix cares most about: a standardized way for a delegator to hand a delegatee a portable, attenuated, chain-verifiable grant.

References

Cedar language and engine

- Cedar project site — positioning, project overview, CNCF status.
- Cedar documentation — primary language reference.
    - Basic Cedar syntax — permit/forbid effects, scope, when/unless conditions, and the forbid-overrides-permit / default-deny combining rules (scorecard rows 2, 5, 7).
    - Terms & concepts — entities, typed UIDs, the entity hierarchy, static vs. template-linked policies, RBAC/ABAC/ReBAC framing (scorecard rows 1, 3).
    - Policy templates and Roles with policy templates — `?principal` / `?resource` slots, template-linked policies as a runtime grant mechanism, and their coupling to user life-cycle management (row 3, the "delegation as template instantiation" argument).
- Cedar: A New Language for Expressive, Fast, Safe, and Analyzable Authorization (extended version, arXiv 2403.04651) — the design paper; basis for the non-Turing-complete / formally-verified / analyzable claims and the two-slot template restriction (scorecard row 2 analyzability, scorecard row 5 offline subset proof).
- cedar-policy/cedar (GitHub) and cedar_policy crate (docs.rs) — reference implementation; entity store model, local in-process evaluation, validator/typechecking.

Janssen Project Cedarling (the deployed embedding)

- Cedarling overview / getting started — embeddable PDP on the Rust Cedar engine, WASM/mobile/server bindings, Token-Based Access Control, local edge evaluation (scorecard rows 4, 6, offline nice-to-have).
- Authorization using Cedarling — derivation of separate User (id_token) and Workload (access_token) principals, the "person AND workload must be authorized" decision logic, JWT validation steps (scorecard row 1, PoP nice-to-have).
- Cedarling getting started — multi-issuer authorization — `authorize_multi_issuer` and `authorize_unsigned` methods; multiple trusted issuers in one request (scorecard rows 4, 7).
- Policy Store — schema + policies + trusted issuers bundle, claim mapping, role mapping; loaded as static JSON or fetched over HTTPS / from a Lock Server (scorecard rows 4, 6).

Agentic case study

- Clawdrey Hepburn — autonomous AI agent using an embedded Cedar engine as its action-space boundary.
    - Why I Built a Policy Engine Before I Built a Personality — per-action Cedar checks as enforced (not advisory) boundaries; the "follow only accounts Sarah follows" example (scorecard rows 1, 2).
    - Modeling an Agent's World in Cedar — typed schema modeling the agent, operator, and action space; agent nested as a principal under its human operator (scorecard row 1).

input: AuthZEN

Chapter: AuthZEN

Overview

This chapter evaluates AuthZEN against the seven mandatory criteria and the three additional considerations defined in the main chapter.

For the purposes of this assessment, we distinguish between the AuthZEN Core Specification and the broader AuthZEN ecosystem. we refer to as the Core in this Chapter.

The term Core Specification refers to the AuthZEN Authorization API 1.0 (hereafter referred to as “Core”), which defines the normative interoperability model between a Policy Enforcement Point (PEP) and a Policy Decision Point (PDP). As the primary standards-track specification and the foundation for runtime authorization interoperability, it serves as the baseline for evaluating AuthZEN against each criterion.

The broader AuthZEN ecosystem currently includes three complementary specifications:

It is important to note that these specifications are not all at the same level of maturity. AuthZEN Authorization API 1.0 is a Final Specification and serves as the normative foundation for this assessment. By contrast, COAZ and COAZ-MCP are currently Draft 1 specifications, while the AuthZEN Policy Store Format remains a working draft (GitHub Repo). Accordingly, any assessment of the broader AuthZEN ecosystem should be understood as including current draft specifications in addition to the finalized Core specification.

- AuthZEN Policy Store Format (hereafter referred to as “Policy Store”), which defines a portable and PDP-neutral packaging format for policy artifacts, schemas, trusted issuers, metadata, and related configuration.
- COAZ: A Framework for Mapping Information Models to AuthZEN Authorization Requests (hereafter referred to as “COAZ”), which defines a protocol-neutral framework for translating external protocol interactions into AuthZEN Subject-Action-Resource-Context (SARC) authorization requests.
- COAZ-MCP: COAZ Binding for the Model Context Protocol (hereafter referred to as “COAZ-MCP”), which applies the COAZ model specifically to the Model Context Protocol (MCP), enabling parameter-aware authorization decisions for AI agents, MCP gateways, and MCP servers.

The evaluation therefore follows a layered approach. Each criterion is first assessed against the capabilities provided by the Core specification alone. Where relevant, the analysis then identifies how Policy Store, COAZ, or COAZ-MCP specifications materially strengthen, clarify, extend, or operationalize those capabilities. This distinction prevents ecosystem features from being incorrectly attributed to the core protocol while still allowing AuthZEN to be evaluated as a practical authorization framework for agentic systems.

Its central data model is a runtime authorization request expressed in terms of Subject, Action, Resource, and Context (SARC). The core specification defines single Access Evaluation and multiple Access Evaluations endpoints together with Search operations for discovering

permissible resources, subjects, or actions. At the same time, it explicitly excludes policy language, architecture, policy state management aspects of a PDP and API authentication from its scope. Accordingly, this chapter distinguishes functionalities that are intrinsic to the AuthZEN protocol from functionalities supplied by the PDP, policy engine, identity system, or another surrounding layer.

The Policy Store addresses the policy-management side of the architecture by defining a PDP-neutral package, the bundle of policy files in a directory structure, for co-locating policies, schema, default entities, trusted token issuers, and supporting metadata. Also, this specification defines Policy Store API to manage the package. Its purpose is portability, versioning, auditability, and interoperability of policy artifacts across PDPs and tooling. Importantly, it standardizes the packaging and management format rather than the underlying semantics of a particular policy language or engine, thereby preserving implementation flexibility within PDPs while enabling interoperable policy distribution and lifecycle management.

COAZ addresses the boundary between external protocols and AuthZEN. It defines a protocol-neutral framework for declaratively mapping the inputs of an incoming operation into a SARC authorization request. Values may be fixed or computed from operation inputs, normally using Common Expression Language (CEL), allowing authorization decisions to reflect the actual parameters and context of an operation rather than only a caller identity. COAZ itself is protocol-agnostic, while protocol-specific bindings such as COAZ-MCP define how a particular protocol is projected into the SARC model.

Finally, COAZ-MCP applies that framework specifically to the Model Context Protocol. It maps MCP JSON-RPC operations into AuthZEN evaluations, providing default mappings for MCP methods and allowing tool-specific mappings where greater precision is required. This is particularly relevant to AI-agent authorization because it enables fine-grained, parameter-level policy evaluation at MCP gateways and servers. In the same way, future bindings could apply COAZ to other protocols, such as HTTP APIs, OpenAPI-described routes, gRPC services, or additional agent frameworks.

Many of the entries in the scorecard are marked as No, but this does not imply that AuthZEN is poorly suited to agentic authorization. Rather, they reflect the fact that AuthZEN addresses a different layer of the problem. Its primary role is to standardize runtime access decisions and PEP/PDP interoperability. Delegation chains, attenuation, revocation, and related capabilities remain the responsibility of capability systems or delegation model provided elsewhere in the architecture. As explained in the main section, an agent's required permissions cannot be fully known in advance, making least privilege dynamic. AuthZEN's access evaluation and search model is directly relevant to that problem because decisions can be made against the concrete action, resource, and context at execution time. Therefore, rather than representing a limitation, this can be considered a distinctive strength of AuthZEN, enabling it to be used without violating the requirements of agentic systems.

Scorecard

#	Requirement	Verdict	Notes
1	**Accountable** (agent vs. principal/operator)	Partial	AuthZEN core can identify a subject but does not standardize relationship between the responsible-principal and the agent. COAZ-MCP explicitly separates human subject and agent context, which improves agent accountability. However, it does not establish a delegation proof or legal-operator binding.
2	**Resistant to confused deputy** (Composable: combine sources in one request)	Partial (deployment-dependent)	AuthZEN evaluates explicit Subject-Action-Resource tuples and supports resource-specific authorization decisions. COAZ and COAZ-MCP further bind authorization to protocol inputs and trust-anchored identities. However, the ecosystem lacks delegation artifacts and end-to-end cryptographic delegation chains, leaving confused-deputy resistance dependent on correct PEP behavior and deployment architecture.
3	**Represent authorization policies** (embed/link; “undelegatable”, re-delegation rules, advisory “please”)	No for core (out of scope) Partial with ecosystem	Policy language, architecture, and state management of the PDP are outside the scope of AuthZEN. The ecosystem improves policy-artifact portability and request-binding interoperability through Policy Store, COAZ, and COAZ-MCP. However, authorization-policy semantics remain policy-engine-specific and are not inherently interoperable across policy engines or organization.
4	**Chainable**	No	Verdict is "No" because it is not a delegation framework. Not because it is incompatible with delegation frameworks. Neither the core data model nor COAZ-MCP contains a normative parent grant, delegator, delegatee, delegation edge, parent capability, or proof-chain primitive. COAZ-MCP improves subject/agent attribution, but not multi-hop delegation. Access Evaluations support request batching (boxcarring) but not chaining. Each evaluation is processed independently and cannot depend on the outcome of another evaluation in the same request.

5	**Cross-organizational / locally verifiable**	Partial (deployment-dependent)	Enable cross-vendor permission request/decision interoperability. Cross-organizational use is possible through a trusted remote PDP, but AuthZEN does not standardize policy exchange or provide locally verifiable proof of delegated authority. The receiving organization relies on the remote PDP's decision.
6	**Attenuated**	No	The core API has no native delegation or attenuation model. The broader ecosystem also does not establish that downstream authority is a non-widening subset of upstream authority. Any attenuation must therefore be implemented by the underlying policy engine and data model, not by AuthZEN itself.
7	**Self-revocable** (holder or anyone in chain)	No	No native self-revocation of delegated authority. Revocation is a responsibility of surrounding policy/token/delegation systems. Policy changes can revoke access, but this is different from delegation-chain self-revocation by the delegator or holder.
+	Authentication / Proof of Possession	No (out of scope)	AuthZEN relies on external authentication mechanisms and does not define holder proof-of-possession for delegated authority.
+	Privacy of delegation chain	No (undefined)	There is no standardized delegation chain to disclose or hide. An implementation could store one entirely PDP-side and thereby hide it from the agent, but that is an architectural consequence, not an AuthZEN feature.
+	Offline-capable	No	AuthZEN is transport-agnostic and a PDP can be deployed locally. However, "Offline" in this context means the ability to create, carry, present, and verify delegated authority without contacting a PDP. Because AuthZEN defines neither a native delegation artifact nor an offline-verifiable delegation primitive, its Offline-capable verdict should be No.

Detailed Evaluation

1. Accountable (agent vs. principal/operator)

The Core can identify a Subject, defined as a "user or machine principal," through mandatory type and id fields and optional properties. An Access Evaluation API contains one and only one

Subject. While that Subject may represent either a user or a machine principal, the specification provides no standard way to model additional Subjects, distinguish an acting agent from an accountable principal/operator, or express a standardized relationship between them. Context could carry operator, delegator, or chain information, but their semantics would be implementation-specific rather than interoperable. Therefore, an auditor cannot reliably reconstruct the accountable principal or hop-by-hop delegation solely from a conformant AuthZEN request. The specification also excludes policy architecture, state management, and API authentication from its scope.

The result is more favorable when considering the broader ecosystem rather than the Core specification alone.

With COAZ-MCP, the binding explicitly separates the human subject from the acting agent by allowing the human identity to populate subject.id while representing the agent in context.agent. As a result, auditors can determine not only that an agent acted, but also on whose behalf the authorization decision was evaluated. However, the specification does not define delegation relationships or delegation chains.

COAZ strengthens the trustworthiness of this model through mapping controls and trust-anchored fields. Bindings may require critical fields, such as subject identity, to be derived from independently verifiable inputs rather than caller-supplied data. This improves identity provenance and reduces the risk of agent-supplied identity assertions. Nevertheless, COAZ remains an identity-to-authorization mapping framework and does not define delegation relationships or delegation chains.

Thus, the ecosystem improves agent-versus-principal attribution and identity provenance, but it still falls short of providing interoperable delegation provenance and multi-hop accountability.

2. Resistant to confused deputy

AuthZEN's data model provides a structural foundation for confused-deputy resistance because every Access Evaluation request identifies a required Subject, Action, and Resource, with optional Context. Resource represents the target, while Action represents the intended operation and may contain operation parameters. Consequently, AuthZEN can ask “may this subject perform this specific action on this specific resource?” rather than merely checking whether an identity has a generic role. However, the API does not define how the PEP derives these values from the actual invocation. Subject, Resource, and Action attributes are supplied by the PEP, API authentication is out of scope, and enforcement remains the PEP’s responsibility. Therefore, AuthZEN alone neither proves that the Resource/Action matches what the upstream requester designated nor ensures that the authority used for a downstream operation is the authority intended by the requester rather than the deputy's own ambient authority.

COAZ and COAZ-MCP provide procedural binding between an invocation, the constructed authorization request, and the resulting authorization decision, reducing the likelihood that designation is lost and authorization decisions become based on the deputy's ambient authority.

COAZ requires bindings to specify the source and structure of input variables, treats mapping inputs as potentially untrusted, provides a mechanism for trust-anchored fields, and requires mapping inconsistencies or evaluation failures to be treated as mapping errors. COAZ-MCP binds MCP tool names, arguments, resource URIs, token claims, and server audience information into authorization requests. Its trust-anchored subject.id may be verified against the designated subject-identity claim, while declared mappings allow authorization policies to incorporate tool parameters directly.

Taken together, these ecosystem specifications strengthen the preservation of requester intent and reduce the risk that access decisions become detached from the actual operation being performed. However, they still lack a cryptographically transferable designation-and-authority artifact and end-to-end attenuating delegation chain. Consequently, confused-deputy resistance remains partial and deployment-dependent.

3. Represent authorization policies

Core specification does not standardize an authorization-policy representation. The Access Evaluation model provides SARC, but it does not define a standardized mechanism for embedding or referencing authorization policies themselves. Furthermore, the PDP's policy language, architecture, and state management are explicitly outside the scope of the specification. The specification illustrates advice, obligations, reasons, and step-up instructions as possible uses of Decision Context, but explicitly leaves their semantics and format outside the standard. Consequently, AuthZEN does not provide interoperable semantics for advisory rules, re-delegation constraints, or disclosure-governance policies.

Policy Store standardizes policy packaging and interchange, not policy semantics. Policies remain policy-engine-specific. Consequently, policy artifacts can be exchanged and discovered through a common packaging mechanism, but the semantics of delegation, re-delegation rules, undelegatable authority, and other authorization policies are not inherently interoperable across policy engines or organizations, since their meaning and enforcement are still defined by the underlying policy engine and its policy model.

COAZ and COAZ-MCP standardize how operations are mapped to AuthZEN authorization requests (COAZ Mapping, COAZ-MCP Declaring a Mapping), not the semantics of authorization policies. COAZ provides declarative mappings from protocol inputs into AuthZEN SARC requests, while COAZ-MCP allows such mappings to be embedded in MCP tool definitions and enforced by a PEP before execution. Consequently, operations can be bound to authorization requests in a portable and interoperable manner, but the meaning of the policies being evaluated remains defined by the underlying PDP and policy engine rather than by COAZ or AuthZEN.

None of these specifications defines a common authorization-policy language or semantic model. They improve interoperability at the authorization interface layer (requests, responses, and policy artifacts), but not at the policy semantic layer. Consequently, different PDPs can exchange authorization requests and decisions through a common protocol, and policy artifacts

may be shared through common packaging mechanisms, but the meaning and enforcement semantics of those policies remain implementation specific.

4. Chainable

If being chainable is interpreted as preserving delegated authority across multiple hops, AuthZEN and its current ecosystem do not provide a standardized delegation-chain model. Delegated authority remains external to the AuthZEN interoperability model. The same observation largely applies to provenance reconstruction. However, AuthZEN itself should not be interpreted as an impersonation-oriented model. Although it does not solve multi-hop delegation, neither does it force workflows into impersonation. Instead, AuthZEN can consume and evaluate externally provided delegation-related context as part of a delegated-authority workflow.

The Access Evaluations API should likewise not be interpreted as delegation chaining. It supports batching (boxcarring) of authorization requests, but each evaluation remains an independent authorization decision. Multiple evaluations do not create, transfer, attenuate, or propagate authority between actors. Consequently, request batching is fundamentally different from a genuine delegation chain and should not receive chainability credit.

The ecosystem specifications improve interoperability but do not materially change this conclusion. Policy Store, COAZ, and COAZ-MCP improve portability, interoperability, and attribution, but none of them define delegated authority as a first-class, verifiable artifact across multiple hops. Nevertheless, like AuthZEN itself, they do not inherently force delegated-authority workflows into impersonation.

AuthZEN Issue #416 "[Feature Request] Add 'delegation_chain' to Subject for AI Agent and Multi-Actor Scenarios" further illustrates this point, which proposes introducing a delegation_chain structure for AI-agent scenarios. The proposal explicitly argues that the current subject model cannot adequately represent multi-hop delegation chains involving users, agents, and downstream services. This should not be interpreted as existing support. Instead, it demonstrates that delegation provenance and multi-hop authority flow are recognized gaps in the current model.

Accordingly, if chainability is viewed as a property of delegated authority itself rather than authorization interoperability, AuthZEN should still receive No. Nevertheless, the reason is not that AuthZEN prevents delegation chains or requires impersonation. AuthZEN deliberately operates at a different architectural layer. It can evaluate and enforce authorization decisions informed by delegated-authority context, but the delegation chain itself, including attenuation, provenance, chain verification, and revocation semantics, must be supplied by another system. AuthZEN is therefore best characterized as a consumer of delegation chains rather than a provider of delegation chains.

5. Cross-organizational / locally verifiable

The Core enables authorization interoperability across organizational boundaries and the ecosystem improves it, but participating organizations must separately establish trust in the

relevant token issuers, identity claims, and PDPs. (COAZ Trust-Anchored fields improve input trustworthiness but do not establish cross-organizational trust.) The Core does not define how such cross-organizational trust is established, and policy semantics remain external to the AuthZEN ecosystem.

The Access Evaluation API returns an access decision, but not the underlying proof needed for another organization to independently validate why that decision was made. Nor do Policy Store, COAZ, or COAZ-MCP define such an authorization-proof format. AuthZEN ultimately relies on an access decision made by a PDP.

Consequently, an organization can operate its own PDP locally, and organizations can integrate their authorization systems across organizational boundaries, but neither outcome constitutes local verification of authorization evidence. The receiving organization still relies on the PDP that made the access decision and on the separately configured identity and token trust infrastructure, rather than independently verifying a transferable delegation or authorization artifact. The criterion is therefore rated Partial.

Cross-organizational deployments can be strengthened by combining AuthZEN with external trust frameworks for issuer trust, identity federation, key discovery, delegation credentials, and shared policy semantics, while continuing to use AuthZEN as the interoperable authorization layer connecting otherwise independent trust and policy domains.

6. Attenuated

The Core alone can express narrowly scoped, context-sensitive access questions through SARC, making it well suited to runtime least-privilege enforcement. However, as with the previous sections, given the scope of the specification, neither monotonic attenuation nor the prevention of authority widening across delegation hops is defined by AuthZEN itself.

Adding the three ecosystem specifications does not improve support for attenuation. As discussed in the previous sections, the Policy Store does not establish parent-child delegation semantics or a monotonic subset invariant. COAZ strengthens the trustworthy construction of fine-grained authorization inputs, but it explicitly preserves AuthZEN request semantics rather than defining delegation semantics. COAZ-MCP extends this mapping framework to MCP, and its controls can enforce constraints supplied by policy. However, none of these specifications establishes that a child delegation cannot exceed the authority of its parent.

Thus, AuthZEN provides strong runtime enforcement for attenuated authority defined and validated elsewhere, but it does not provide native delegation attenuation.

7. Self-revocable

The core specification provides no protocol mechanism by which a holder or an ancestor in a delegation chain can revoke a specific delegated authority, nor does it define any requirement for revocation to cascade to descendants. While changes to PDP policy may cause subsequent evaluations to return false, the specification does not define who is authorized to make such changes or whether that authority derives from participation in a delegation chain.

The specification also defines no revocation-state object, revocation-propagation protocol, cache-invalidation mechanism, network-partition behavior, or maximum revocation latency. Although expiration timestamps or other short-lived authorization evidence may be supplied as contextual input by an implementation, this represents expiration rather than explicit revocation and is not standardized by the API.

The ecosystem does not add delegation-chain self-revocation capabilities. The Policy Store specification allows administrators to upload immutable, versioned Policy Store packages through a POST-only Policy Store API, while existing policy stores cannot be updated or deleted. This supports policy deployment, versioning, and rollback, but not holder- or ancestor-initiated revocation. COAZ defines how protocol-specific information models are mapped into AuthZEN authorization requests, but it introduces neither delegation lineage nor revocation operations. Similarly, COAZ-MCP does not define holder- or ancestor-driven revocation, child-selective revocation, cascading revocation semantics, or guarantees regarding cache consistency, revocation propagation, or network partitions.

8. Authentication / Proof of Possession

Authentication of the Authorization API is explicitly out of scope. While OAuth 2.0 support is RECOMMENDED, OAuth 2.0 deployments are typically based on bearer tokens and therefore do not inherently provide holder proof-of-possession. Achieving proof-of-possession generally requires additional mechanisms such as DPoP, mTLS, sender-constrained tokens, or comparable cryptographic binding techniques. AuthZEN neither mandates nor standardizes any of these mechanisms. Consequently, while AuthZEN can operate in deployments that provide proof-of-possession through external identity and transport layers, the specification itself does not define holder-bound credentials or cryptographic proof-of-possession for delegated authority.

The ecosystem improves the integrity and trustworthiness of identity-related inputs, but it still does not satisfy the proof-of-possession criterion. COAZ can designate trust-anchored fields that must be derived from trusted inputs or verified by the PEP. COAZ-MCP uses decoded JWT OAuth access-token claims and recommends anchoring subject.id to the designated subject-identity claim. It also keeps agent identity separately in context.agent. These mechanisms reduce the risk of identity-claim substitution, but they do not require sender-constrained tokens, a signed invocation, or proof that the agent possesses a private key bound to delegated authority. The Policy Store packages trusted-issuer configuration and policy artifacts, but it likewise does not establish holder proof-of-possession for delegated authority.

9. Privacy of delegation chain

AuthZEN API 1.0 does not satisfy this criterion because the specification does not define a delegation-chain representation and therefore provides no standardized mechanism for concealing, selectively disclosing, or verifying delegation chains. While an implementation could include delegation-related information in custom properties or context fields, the semantics and

privacy characteristics of such information remain implementation-specific rather than standardized by AuthZEN.

Including the ecosystem specifications does not change this conclusion. Neither COAZ nor COAZ-MCP defines a multi-hop delegation chain or any mechanism for hiding selected hops from the MCP client or intermediary. Any delegation-chain privacy that may be achieved through a particular deployment architecture is implementation-specific rather than standardized by the AuthZEN ecosystem.

Accordingly, AuthZEN does not satisfy this criterion. More precisely, the property is undefined because no delegation-chain representation exists within the specification.

10. Offline-capable

The Core does not satisfy this criterion because the specification does not define a self-contained delegated-authority artifact that can be created, delegated, presented, and independently verified without PDP involvement. The normal AuthZEN model is the evaluation of an authorization request by a PDP at runtime. Although the specification is transport-agnostic and permits locally deployed or embedded PDPs, local communication is not equivalent to offline delegation. The ability to evaluate authorization decisions without an external network connection should not be confused with the ability to create, transfer, and verify delegated authority offline.

The same conclusion applies when the ecosystem specifications are considered. Neither Policy Store, COAZ, nor COAZ-MCP defines a self-contained delegated-authority artifact that can be presented and independently verified outside the PDP evaluation model. Consequently, the ecosystem provides no standardized basis for offline delegation. While some deployments may reduce network dependencies through locally available policy artifacts or locally deployed PDPs, runtime authorization remains dependent on a PDP or policy engine rather than a self-verifying delegation artifact.

More precisely, locally deployed or disconnected PDPs may support offline policy evaluation, but they do not provide offline delegation.

AuthZEN-specific Considerations

1. Treat Policy Store portability carefully

AuthZEN standardizes the container and deployment unit for policies, schemas, entities, and issuer configuration, enabling these artifacts to be packaged, exchanged, and reused across implementations. However, Policy Store portability should not be confused with delegation interoperability. While AuthZEN standardizes the packaging and exchange of policy artifacts, the semantics of delegation, attenuation, and revocation remain specific to the underlying capability system implementation and architecture. Consequently, a policy store may be portable between systems, while the resulting delegation behavior may differ across authorization engines and deployments.

2. Search results should not be interpreted as proof of capability or delegation

AuthZEN Search is designed for discovery rather than for conveying authority. The specification states that "Search APIs provide lists of resources, subjects or actions that would be allowed access." Search results indicate what may be permitted under the PDP's current state, but they do not by themselves constitute proof of delegated authority, capability possession, or authorization grants.

3. Search can create an information-disclosure risk

Reverse-query APIs inherently expose information derived from the current state of the authorization system. If made available to insufficiently trusted callers, repeated or carefully crafted searches may reveal the existence of resources, access relationships, or characteristics of effective authorization policies. While Search can improve usability and discovery, it can also increase the risk of information disclosure. Appropriate authentication, query scoping, rate limiting, response minimization, and careful handling of diagnostic information should therefore be considered.

4. Policy Store administration

Policy Store specification does not define a normative authorization model for administration of the Policy Store itself. Section 13, “Security Considerations,” indicates that security, authorization, and operational controls of Policy Store itself are implementation responsibilities rather than standardized Policy Store behavior. Specifically, the specification does not prescribe who may create, modify, approve, publish, or delete policy bundles, nor does it define administrative roles, delegation models, or approval workflows for policy governance. These responsibilities are left to the implementing environment and its operational controls. As a result, organizations may adopt different approaches, ranging from repository-based access controls and CI/CD approval processes to dedicated PAP solutions. This separation preserves implementation flexibility, but it also means that governance, segregation of duties, and administrative access control for the Policy Store remain implementation-specific rather than standardized by AuthZEN.

5. Decision Interoperability vs. Policy Interoperability

The Authorization API standardizes how a PEP requests and receives access decisions from a PDP, but it does not standardize policy languages, policy semantics, or policy execution models. A conformant PDP may use Cedar, Rego, Zanzibar-style relationships, or other policy systems, while exposing the same AuthZEN interface. AuthZEN achieves interoperability through interoperable authorization requests and access decisions, rather than through interoperability of policy semantics. One organization can consume access decisions produced by another organization's PDP, but AuthZEN does not require PDPs to share, exchange, interpret, or enforce one another's policies.

Summary

AuthZEN can serve as a strong runtime authorization and policy-decision layer within an agentic authorization stack. Its primary value lies in standardizing authorization decisions and interoperability between authorization components, while supporting emerging AI-agent and tool-invocation scenarios through initiatives such as COAZ and COAZ-MCP.

However, AuthZEN is not a delegated-authority system. Delegation semantics, including delegation proof, attenuation, and revocation, remain outside the scope of Authorization API 1.0 and are provided by other components of the overall architecture. Likewise, policy representation is intentionally left to the underlying PDP implementation.

Understanding these boundaries is important when comparing AuthZEN with policy frameworks such as Cedar and delegation-focused approaches such as UCAN and zCAP.

References

- OpenID AuthZEN Authorization API 1.0 Specification
- OpenID AuthZEN Policy Store Format Specification
- COAZ: A Framework for Mapping Information Models to AuthZEN Authorization Requests - Draft 1
- COAZ-MCP: COAZ Binding for the Model Context Protocol - Draft 1
- AuthZEN GitHub Issue #416 “[Feature Request] Add 'delegation_chain' to Subject for AI Agent and Multi-Actor Scenarios”
- OpenID AuthZEN Meeting Notes
- AuthZEN Interop

input: VCs for Authorization

How would you model these as a VC?

INCLUDE KYA-OS as an example?

* First thing, figure out what to model as the subject.
    - is the permission itself the subject?
    - is the agent the subject?
* which fields / claims will refer to: agent, principal, resource, actions, caveats, ~~revocation~~.
* **how do you do proof chains?**

1) root permission (for example, human principal)
  2) delegates to: Agent A (personal data manager), has READ and WRITE
    3) delegates to: Agent B (backup service), READ only

zCap example:
  1) root zcap:

```
{
  "id": "urn:zcap:root:https%3A%2F%2Fexample.com%2Fdocuments",
  "controller": "did:key:z6Mkfeco2NSEPeFV3DkjNSabaCza1EoS3CmqLb1eJ5BriiaR",
  "invocationTarget": "https://example.com/documents"
}
```

  notice: no set of actions -- root zcaps have "admin" rights / all permissions

  2) first hop delegation

```
  "id": "urn:zcap:delegated:z9gLKoFmKHwhxCzmo91Ywnh",
  "parentCapability": "urn:zcap:root:https%3A%2F%2Fexample.com%2Fdocuments",
  "invocationTarget": "https://example.com/documents",  ← resource
  "controller": "did:key:z6MknBxrctS4KsfiBsEaXsfnrnfNYTvDjVpLYYUAN6PX2EfG", <- agent
  "expires": "2022-11-28T20:53:06Z",
  "allowedAction": ["read", "write"],


  "proof": { <- this is a signature
    "type": "Ed25519Signature2020",
    "created": "2021-11-28T20:53:06Z",
    "verificationMethod":
"did:key:z6Mkfeco2NSEPeFV3DkjNSabaCza1EoS3CmqLb1eJ5BriiaR#z6Mkfeco2NSEPeFV3DkjNSabaCz
a1EoS3CmqLb1eJ5BriiaR",
    "proofPurpose": "capabilityDelegation",
    "capabilityChain": [
      "urn:zcap:root:https%3A%2F%2Fexample.com%2Fdocuments"
    ],
```

```
    "proofValue":
"z244yxzRuFMyGfK85QcE6UewEZ3JpGDDTCvBKuxNiwdnxF3AmsSAoVYTBPLvFpYV7SeeWB4tUBGMGTF7pka6
xR3av"
```

```
  3) second hop: from general data agent to Backup agent
  "id": "urn:zcap:delegated:z12345",
  "parentCapability": "urn:zcap:delegated:z9gLKoFmKHwhxCzmo91Ywnh", <- link to step 2
  "invocationTarget": "https://example.com/documents",  ← resource
  "controller": "did:key:<did of the backup agent>", <- agent
  "allowedAction": ["read"],  <- attenuated (narrower than parent)
```

```
  "proof": {
    "type": "Ed25519Signature2020",
    "created": "2021-11-28T20:53:06Z",
    "verificationMethod":
"did:key:z6Mkfeco2NSEPeFV3DkjNSabaCza1EoS3CmqLb1eJ5BriiaR#z6Mkfeco2NSEPeFV3DkjNSabaCz
a1EoS3CmqLb1eJ5BriiaR",
    "proofPurpose": "capabilityDelegation",
    "capabilityChain": [
      "urn:zcap:root:https%3A%2F%2Fexample.com%2Fdocuments",
      "urn:zcap:delegated:z9gLKoFmKHwhxCzmo91Ywnh"  <- by id OR
           < copy and paste the whole parent capability>
    ],
    "proofValue":
"z244yxzRuFMyGfK85QcE6UewEZ3JpGDDTCvBKuxNiwdnxF3AmsSAoVYTBPLvFpYV7SeeWB4tUBGMGTF7pka6
xR3av"
```

TODO: Add the * sub-delegation ability to zCaps spec

VCs

* Challenge: some of the fields used for general claims are confusing in permissions.
    - VCs use a PUBLIC-readable revocation list (or at least, permissioned bistring status lists are not specified), but in Caps, only the resource server needs to read the revocation list, so there's a mismatch there (different privacy requirements)
    - VCs have one half of the Cap protocol -- they have the delegation, but have no notion of **invocation** (there are Verifiable Presentations for proof of possession, but they'd need extra specifications on how to use them to invoke permission)

Section: Gotchas and Things To Watch Out For
example: (Alan) don't sign the request and the invocation separately.

* Future Q: What do you do when you don't know who the agent is supposed to be?

Option 1: VC - Agent as subject?

iss: principal's DID
type: ["VerifiableCredential",
"**PermissionCredential**"]
validFrom / validTo: <not before / exp>
credentialSubject:
id: agent's DID <-`controller`
hasPermission: [
{
id: <optional permission id>
resource (obj), <- `invocationTarget`
action: [arr of strings or objs],
policy: link to policies,
caveats here?
}
],
agentPlatform: <claims go here>
operator/principal: <claims go here>

Option 2: VC - Permission as subject?

iss: principal's DID
type: ["VerifiableCredential",
"**PermissionCredential**"]
validFrom / validTo: <not before / exp>
credentialSubject:
id: <optional permission id>
agent: {
id: <agent DID>
agentPlatform: <claims>
operator/principal: <claims>
}
resource (obj),
action: [arr of strings or objs],
policy: link to policies,
caveats here?

```
  "id":
"urn:zcap:delegated:z9gLKoFmKHwhxCzmo91Y
wnh",
  "parentCapability":
"urn:zcap:root:https%3A%2F%2Fexample.com
%2Fdocuments",
  "invocationTarget":
"https://example.com/documents",  ←
resource
  "controller":
"did:key:z6MknBxrctS4KsfiBsEaXsfnrnfNYTv
DjVpLYYUAN6PX2EfG", <- agent
  "expires": "2022-11-28T20:53:06Z",
  "allowedAction": ["read", "write"],
```

input: Other Certificate Capabiiities

Chapter: Other Certificate Capabilities

Capabilities for Intelligent Agents are coming into fashion, and certificate capabilities are leading the way. Other chapters in this report detail a few of them, but there is a growing number of such projects, too many for the detailed descriptions of the leading candidates in this document. This chapter provides a brief description of many of these other systems.

French Toast

A lot like zcaps but with a limited delegation depth. “Implementations MUST impose a maximum chain depth to prevent denial-of-service. A depth limit of 10 is RECOMMENDED.”

Provenance Identity Continuity (PIC)

Nicola’s project. At release 0.1. Proof of continuity to prevent use for another purpose.

Cedarling

Web Authorization Protocol

Replicable Capability System

Transaction Tokens

SPKI/SDSI

Not a complete spec. Includes a do-not-delegate bit.

Macaroons

Bearer tokens. Hash-based instead of signed,

Biscuits

Bearer tokens. Datalog policies.

Agent Power of Attorney

Aries Chained Credentials

VCALM

Kepler

I saw this project at IIW before it was picked up by Spruce. These guys know what they're talking about.

Transaction Tokens

OAuth based.

Zebra Copy

SAML assertions. Not a spec.

Meadowcap (Willow Protocol)

- See the food section of this

NOT Certificate Capabilities and out-of-scope but sometimes mentioned:

- OCapN (platform caps, not cert caps; cap-aware transport; see Kenton Varda's upstream transport-protocol work for Cloudflare)

Notes

Task Force Roadmap

- ~~(DONE) Resolve discussion comments~~
- ~~(DONE) Explicitly state the list of technical requirements for delegated authorization specifications~~
    - Marc Siegler's 7 properties
    - needs to be cross-organization / cross-domain
    - ability to explicitly state the delegation and governance policy
        - can the policy be changed after authorization is granted?
        - risk adaptive -- external conditions
        - policy itself is dynamically evaluated (for example, geofencing)
    - Data Model vs Protocol.
        - need the ability for Agents to ask for additional permission as they perform their work
- (IN PROGRESS) Review existing Delegated Authorization specs, with those requirements
- Do a Gap Analysis (are the current specs missing anything, wrt those reqs)
- If necessary, work on specs that address those gaps

TODO: Section discussing advantages/disadvantages of Opaque Token systems vs Certificate systems family tree

- [ ] Alan to get ball rolling with a short freewrite

TODO: Victor Lu mentioned we may be out-of-date looking at last month's MCP/OAuth2.1 specs… should we invite a speaker from AIIF (what's this? Or do you mean AIIM?) to give us an inside perspective on recent and near-future events?

TODO: Pull in the categories from https://docs.google.com/document/d/1Onxx6Hmt0o5-kII6lJ1Y2ZtDofKzT4BQLrtKnWiPYp4/edit?tab=t.0#heading=h.ugfz0wfgh7jc into the introduction of the Checklist items

Specs Adjacent or Useful to Capability Systems

TODO ^

Common Misconceptions in Capability Systems

- "Alice delegates to Bob, who delegates to Charlie -- this does not mean Alice knows who Charlie is."

Gap Analysis: are there features/topics missing from the existing specs that this Task Force may need to get involved in?

Alan: need to take another look at UCAN spec, check on their Revocation and Invocation functionality.

Alan: note that in the UCAN case, they're modeling the ability to revoke as an explicit separate permission "ucan/revoke". This is the preferred way to do it, we need to bring this up to the zCap spec.

TODO: Dmitri to raise this issue on the zCap spec

TODO: Also, raise an issue about implicit attenuation, and the `collection/*` problem

TODO: (possible gap in zCaps and UCANs?) For Auditability and Provenance, it's often required to know who the **responsible** human / **operator** is, for a given delegation.

- are the delegation certificates the place for it? or the invocation?
- can you store the history / record of the responsible party, and just link to it?

Alan: also, for UCANs (and zCaps when they have the '*' feature), make sure that it cannot be invoked during invocation. An invocation MUST be on a specific resource.

Checklist Matrix (to evaluate specifications)

Context/Assumptions: We're evaluating authorization / access control specifications (data model primarily, but also protocol). Validity check must be done at the time of request (by a component trusted by the resource provider).

1. **Accountable** - Spec must be able to differentiate between the agent and its principal operator (legal entity)
2. Spec must be able to represent the authorization **policies** (either by embedding or linking to them)
    a. This includes examples like "do not delegate further" or “this permission is undelegatable”
    b. This includes re-delegation policies
    c. From Alan: **Please** - A request containing desired rules for redelegating but not refusing to honor violations.
    d. "who gets to know what is a governance topic"
3. **Chainable**: Authorization must be able to be delegated: At root, authorization is delegated from principal/operator to agent. But also, agents must be able to delegate to further subsystems or other agents.
4. Spec must be able to fulfil **cross-organizational** use cases (validation checks are independently verifiable, locally verifiable). (To put it another way, different

organizations/groups should be able to express authorization policies without having accounts on each other's IAM systems).
5. Delegation must be able to be **attenuated** (share only a part of the permissions).
6. Authorization must be **self-revocable** (HOLDER (or somebody in the delegation chain) must be able to revoke a delegated permission (modulo network partitions))
7. ~~**Composable** - You must be able to use delegations from different sources in the same request.~~
    a. Counter: some RBAC systems only allow an agent to be in a single role; this violates that requirement; authorization must be composable.
8. Susceptible to **Confused Deputy**?

Not required but nice to have:

- **Authentication/Proof of Possession:**
- **Privacy of Delegation chain:** can you hide delegates in the proof chain from agent / token carrier (the principal knows the full story, the verifier knows the full story, but the intermediate agent doesn't have to know)
- **Offline-capable** (create tokens offline, delegate offline, use/present offline)
    - Agentic Use cases: most of them online, but some are offline-capable to (local machine, or local LAN only agentic use cases)
    - Also, offline-capable parts of the spec can offer efficiency advantages: for example, if you can delegate without having to contact the resource server, that offers a significant speedup / advantage.
    - **TODO: Offline/online is too binary.** Instead, we need to cover:
        matrix of operations (create permission, delegate, evaluate, revoke, etc)
        which of the parties are *reachable*, and which parts can be cached

Specs to be Evaluated

Rough Specification Categories - See Alan's [Token Types for Attenuated Delegation]

1. Opaque tokens (e.g. OAuth2)
2. Digital Certificate Systems
3. Object references

Q: Where do specs like AP2 fall?
A: Not quite a capability system, just carry payment. The payment system makes the decision. So, they do carry intent/policy, but it's not a permission. Not generalizable -- requires a mutually trusted 3rd party between buyer and merchant.

Alan: "Connection-based agent authentication is another option." (or WIMSE-esq workflow identities, or other “external” (to agent) identifiers)

General Data Model (In Common)

- "agent" - how do you identify the agent? (a DID or a key descriptor, most often)
- "principal" - who is this on behalf of?
- "what resource" - what's the resource the permission is given for
- "actions" - what actions/operations are allowed
- "caveats" / "conditions" - additional restrictions and conditions
- "delegation chain" - can this permission be further delegated
- "protocol" - how do you request this permission?
    - also: "**revocation**" (though revocation has a data model aspect as well)
- "invocation" - how you prove cryptographic key possession, and how you *invoke* a permission for a given resource
-
- **OUT OF SCOPE FOR NOW**
    - “Intent” - too big/cross-cutting to model as a simple property (may be more of an architectural consideration, i.e. how PIC-like/e2e is the information model)
    - that said, **Intent** is going to be critical, so worth defining an extra field as an extension point
    - add a section discussing the tension between -- intent is important to agent permission systems, but might not belong in the capability itself. (meaning, we COULD add an optional extension point property for it. but SHOULD we?)

Specs Overview

	OAuth2.1	AuthZEN	GNAP	zCap	UCANs v1	AAuth
Agent	client_id	n/a!	**client** (rich object)	**controller** (DID)	Delegation: **sub,** Invocation: **iss**	
Principal	**sub**	**subject**	**subject**	implicit (in delegation chain)	Delegation: iss, Invocation: sub	
invocation / key binding?	Optional, via DPoP	no	Yes (DPoP, HTTPSig, MTLS)	Yes, HTTPSig + Invocation	Yes (its own mechanism); keys	

				header	taken from DIDs	
Resources	**scope** OR **RAR** (structured scope). see also: htm/htu claims	**resource** and **context**	**location s** and **datatype s**	**invocatio nTarget** (url or object)	Specified in **args**	
Actions	**scope** (also htm)	**actions**	**actions**	**allowedA ction**	**cmd**	
Caveats / add'l policy	n/a	**context**	?	* (implicit in **invocatio nTarget** structure)	? (folded into **args**, maybe **meta** ?)	
Delegation Chain	* (only via Token exchg)	no	no	Yes! via proof chains	Yes (ucan proof chains)	
Protocol (how do I request a capability?)	OAuth2.1 itself	AuthZEN	GNAP	Out of scope (uses VCALM, GNAP etc)	? (come back to it)	
Revocation	RFC7009	n/a	n/a	Yes (out of band, RS)	Yes (out of band, RS). (also, have a revocation capability (to revoke other caps))	

Division of Labor 💪

- (Dmitri - chapter) **zCaps** - see Chapter: zCaps (Authorization Capabilities)

- (Alan - chapter) **UCANs**
- AuthZen - Sachio?
- (Makki - chapter) **OAuth 2.1 ("the OAuth family")**
- AAuth https://datatracker.ietf.org/doc/html/draft-hardt-oauth-aauth-protocol-01
    - Deb? Maybe w/help from BF?
- (coordinate with KYA-OS team on chapter) **KYA-OS** (now renamed to (pronounced "chaos"))
- (Agne - chapter) **Cedar**: https://www.cedarpolicy.com/en + https://docs.jans.io/stable/
    - explore delegation ability of Cedar. (identity based)
    - see also: Clawdrey Hepburn https://clawdrey.com/

1. https://agntcy.org/ from CISCO
2. https://github.com/ciamshrek/french-toast-jwt
3. W3C VCs
4. AAuth https://datatracker.ietf.org/doc/html/draft-hardt-oauth-aauth-protocol-01
5. * zcap-ld: https://w3c-ccg.github.io/zcap-spec/
6. * UCAN: https://ucan.xyz/specification/
    a. Warning: Weird serialization!
7. * MCP-i (now renamed to KYA-OS (pronounced "chaos"))
8. * GNAP (Grant Negotiation and Authorization Protocol)
    a. see especially new draft: https://github.com/brandnbyr/draft-beyer-agent-identity
9. * SPKI: https://datatracker.ietf.org/doc/html/rfc2693
10. * Zebra Copy: https://shiftleft.com/mirrors/www.hpl.hp.com/techreports/2007/HPL-2007-105.html
11. * Macaroons: https://github.com/rescrv/libmacaroons (Not official; Google never published a spec.)
12. * Waterken: https://shiftleft.com/mirrors//www.hpl.hp.com/techreports/2010/HPL-2010-89.pdf
13. * ZTAuth: https://github.com/ztauthstar
    a.
14. * OAuth 2.1: https://oauth.net/2.1/
15. * Cedar: https://www.cedarpolicy.com/en + https://docs.jans.io/stable/
    a. explore delegation ability of Cedar. (identity based)
    b. see also: Clawdrey Hepburn https://clawdrey.com/
16. * E: http://endojs.org, Spritely
17. * AP2
18. SPIFFE / WIMSE / SCITT
    a. OID-Federation v1.0 spec may be combinable w/SPIFFE
        i. see: https://www.linkedin.com/posts/takahikokawasaki_openid-federation-10-oauth-spiffe-client-activity-7435478895669231616-9wXi
    b. SPIFFE (Secure Production Identity Framework for Everyone) (like DIDs but IETF): https://spiffe.io/docs/latest/spiffe-about/overview/ - a general

purpose system for identifying software systems (such as workloads, agents, etc). defines:
    i. SPIFFE IDs are a Uniform Resource Identifier (URI) which takes the following format: **spiffe://*trust domain*/*workload identifier***
    ii. SVIDs (SPIFFE Verifiable Identity Document) - a signed document (in JWT format or X.509 based). Defines `sub`, `aud`, and `exp` claims.
c. WIMSE (Workload Identity in Multi-System Environments) - Identity specifically for workloads. Explicitly out of scope: Personal identities, Static software identities and provenance, such as software bill of materials (SBOM).
    i. set of complementary specs. Key binding/proof of possession, workload identity docs, etc
    ii. relevant to us: https://datatracker.ietf.org/doc/html/draft-ni-wimse-ai-agent-identity-02
d.

19. KYAPay (not permission system, will take a look at their claims that identify agent, principal, etc)
20. Biscuit: https://www.biscuitsec.org/
21. Archon Protocol P2P challenge/response auth supports VC/VP-based restrictions (https://github.com/archetech/archon/blob/main/doc/WHITEPAPER.md#85-challenge-response-authentication)
22. TODO: look at KERI for chainable proof mechanisms
    a. and the new delegation work
    b. benefit: useful for long lived authorization. (which often is short-lived)

"Classic" OAuth2 / OAuth2.1

- An example of opaque token systems
- Mostly a Protocol spec, with Implied Data Model
    - Token is opaque, which means it's a pointer into a database with arbitrary amounts of fields
    - Implied data model includes:
        - client_id
        - scopes (e.g. "read:email")

        - Not included (in the core spec - this is expanded in further specs):
            - binding to any particular resource or url
- Originally required **pre-registration** (to prevent stolen tokens). This is presenting lots of problem in the agent space.
    - This was partly addressed in later specs - Dynamic Client Registration, DPoP
    - Client Metadata Documents - basically a parallel re-evolution of DIDs
- **Accountable**?
    - Tracks client_id
    - Does not track Principal explicitly, BUT Principal is tracked implicitly (for example, to register a third-party client, you have to be logged in, principal is known)
        - though see efforts like KYAPay IETF draft for tracking principals
- **Policies**?
    - Not explicitly. (Opaque token can index into a backend database that can have policies attached)
    - Scope carries some policy information, but not all
    - Has expiration
- **Chainable**?
    - Core spec: NO. Token is locked to a given client_id (and implicit principal)
    - Extended specs (like **Token Exchange**): Possibly. Further specs (like Token Exchange) allow for an emulation of chainability.
        - However, the delegation chain is opaque, stored on the backend, not carried in the token itself
- **Cross-Organizational**?
    - Core spec (pre-registration): No. (Token is locked to a client_id)
    - Dynamic Client Registration or Client Metadata Documents: Yes, potentially.
- **Attenuated**?
    - Yes* (you ask for a list of scopes, you can attenuate during Token Exchange)
- **Self-Revocable**?
    - Core spec: Not explicitly (though in practice you could revoke an access token from a client, on your control panel)
    - RFC7009 Token Revocation introduced self-revocation ability
- **Composable?**
    - In principle (for example, you can combine multiple access tokens in the Authorization: header, though in practice this is rarely done)

OAuth2/2.1 Limitations:

- Lack of explicit audience- and resource-bindings in the core spec
    - Though this was attempted to be addressed in later specs such as RAR, DPoP, <whatever the audience/url binding one>
- Lack of structured scopes
    - Attempted to be addressed by RAR
- Awkward chainability (via Token Exchange)

Focus use cases

- see also: Alan Karp's Usecases Document
- Cloud storage for agents (I want to give my agent access to my cloud Documents folder)
  - applies to other enterprise services / microservices
  - transitive access usecase, with revocation
  - Debbie: "Delegated authority must be verified at execution time and remain revocable in real time, even across chained agents. Audit alone is insufficient — the green light must be enforced when the action runs."
- A2A Delegation with dynamic revocation
- Financial services / banking: “I need to give my agent permission to make transfers that fit within specified guidelines.”
  - Q: Does this differ from the usecases covered by Google's AP2 protocol?
    - Alex K.: AP2 is one of a handful of payment protocols. Differences & overlap between them.
    - https://ap2-protocol.org/specification/
    - https://stripe.com/blog/agentic-commerce-suite
    - https://developer.visa.com/capabilities/trusted-agent-protocol
    - https://www.mastercard.com/global/en/business/artificial-intelligence/mastercard-agent-pay.html
    - https://github.com/skyfire-xyz/kyapay-ietf-draft
  - SPECIFICALLY: The intersection of Payment + Authorization
- Payment? (we may need simpler / less risky initial use cases)
  - Approving payments workflow?
  - Providing payment to an agent (alongside authorization) and restricting what the payment is for
- travel & hospitality: "I need to give my agent permission to update my hotel check-in reservation if my flight is delayed".
-

- Perform an overview of existing delegated authorization specs
- Perform a Gap Analysis to see if any of the requirements are not covered by existing specs
- Proceed from there

Edits to the Paper

(add edits here)

From Appendix A: adding links to the various technologies

* zcap-ld: https://w3c-ccg.github.io/zcap-spec/
* UCAN: https://ucan.xyz/specification/
* SPKI: https://datatracker.ietf.org/doc/html/rfc2693
* Zebra Copy: https://shiftleft.com/mirrors/www.hpl.hp.com/techreports/2007/HPL-2007-105.html
* Macaroons: https://github.com/rescrv/libmacaroons (Not official; Google never published a spec.)
* Waterken: https://shiftleft.com/mirrors//www.hpl.hp.com/techreports/2010/HPL-2010-89.pdf
* ZTAuth: https://github.com/ztauthstar
* OAuth 2.1: https://oauth.net/2.1/
* Cedar: https://www.cedarpolicy.com/en
* E: http://endojs.org

Please update these links if you know of something better.

Problem Space Report

NOTE!!!

All of the proceeding has been moved to github — to make any changes (even a typo or adding a link!) please do so on github, it will not get noticed here!

Confused Deputy Susceptible Techniques

Confused Deputy Susceptible Techniques

Ambient authority is the main issue.

Cedar (and AuthZen) are authentication/identity-centric, and susceptible to Confused Deputy.

** Put the Cedar and AuthZen chapters here.

These can be useful for *caveats* as a policy language.
But a very different thing from cert-based capabilities.