Identity and trust

Use this page to decide which identity values an AIP implementation may trust. It covers caller authentication, tenant and provider-account selection, credential resolution, operational reads, and connector dispatch. It is for gateway, runt

SECTION	Core Concepts
TYPE	Concept
PROTOCOL	AIP 1.0
SOURCE	docs/concepts/identity-and-trust.md

The central rule is simple: identity carried in an AIP payload is a claim, not proof. A production path first establishes an actor at a transport or signature edge, then enriches that actor from deployment-owned data. The runtime executes with the enriched context and replaces caller-supplied action identity before a handler sees the action.

This page describes the reviewed Rust implementation of AIP 1.0 at source revision `97be86e9efedf07ecf1783b03800f683f107fb04`. A deployment still owns its identity provider, issuer trust, tenant directory, credential store, key lifecycle, and policy. The implementation exposes primitives for those boundaries; it does not create a universal identity federation.

Keep four identity layers separate

Layer	Main type or record	Authority	Purpose
Wire claim	`Principal`, `Envelope.from`, and `Action.identity`	Caller-controlled until verified	Names actors and supplies identity lookup hints
Authenticated actor	`AuthenticatedPrincipal`	Transport authenticator or trusted DID binding	Records the proven principal, scheme, issuer, scopes, time, expiry, and optional credential fingerprint
Trusted enrichment	`ResolvedIdentity`	Deployment-owned identity resolver	Adds verified tenant membership, an opaque credential handle, and sanitized product identity
Execution boundary	`MessageContext` and, for fleets, `RouteAssignment`	Gateway, runtime, and connector control plane	Freezes the context used for authorization, persistence, routing, and handler execution

Parsing moves bytes into typed wire objects. It does not move a value from the first row to the second. Authentication, resolution, and authorization are separate decisions, and each can fail independently.

This separation also prevents identity roles from collapsing into one value. The transport actor can be a service while `human_actor` identifies the person who initiated the work, `acted_on_behalf_of` identifies the represented principal, and `external_user` names the account-local user understood by a provider.

Wire identity describes the requested context

`Principal` is the canonical actor-shaped protocol object. Its `kind` is one of `human`, `agent`, `service`, `tenant`, `customer`, `contact`, or `system`.

Principal field	Meaning	Trust boundary
`id` and `kind`	Stable AIP identity and actor category	Must be bound to authenticated evidence before authorization

Principal field	Meaning	Trust boundary
`display_name`	Human-readable label	Never an authorization key
`trust_domain`	Claimed owning domain	A label until the receiving deployment verifies the immediate peer
`did`	Claimed decentralized identifier	Does not prove key possession by its presence
`external_refs`	Product-local identity references	Require deployment or connector mapping before trusted use
`delegated_authority`	Scoped grants for another principal's work	Authoritative only when retained on a transport-established principal
`auth_context`	Profile or gateway authentication metadata	Its interpretation belongs to the trusted edge, not generic payload parsing

`IdentityContext` describes the identity and account mapping associated with an action or approval. It has eight optional fields:

Field	Intended meaning
`tenant`	Tenant or account isolation boundary
`external_account`	Provider account selected for connector work
`external_user`	Provider-local user associated with the invocation
`human_actor`	Person who initiated or owns the intent
`service_account`	Machine identity performing the operation
`acted_on_behalf_of`	Principal whose authority is being exercised
`credential_ref`	Identifier, issuer, and known scopes for material stored elsewhere
`oauth`	Issuer, client identifier, scopes, expiry, and refresh availability without token material

These fields are useful for routing, audit, policy, and provider mapping, but an incoming `Action.identity` is only a set of lookup hints.

The reviewed static resolver requires exact tenant and claimed-credential matches. When its trusted identity mapping contains account, user, human, or on-behalf-of values, it also rejects conflicting hints. A claim that is not present in the resolver result is discarded rather than copied into runtime state.

Authentication establishes the immediate actor

The runtime representation of a proven caller is `AuthenticatedPrincipal`. It retains the canonical principal together with the authentication scheme, trusted issuer, optional audience, verified scopes, authentication time, optional expiry, and an optional non-secret token or certificate fingerprint. Validation rejects expired authentication and missing required scopes.

The authentication vocabulary contains `did_proof`, `bearer`, `oauth2`, `hmac_webhook`, `mtls`, and `api_key`. This enum is not an assurance that every transport binding implements every scheme. Each network edge must verify its own token, certificate, webhook, or peer credential before calling the trusted gateway entrypoint.

The reviewed gateway has three relevant ingress patterns:

1. Native signed ingress verifies the Ed25519 signature over the canonical

envelope, obtains a `did:key` verifier, looks up that DID in the trusted signer map, and checks that `Envelope.from.id` and `kind` match the bound principal.

2. An already authenticated in-process edge supplies a principal, or supplies

a complete `AuthenticatedPrincipal` plus verified tenant and credential state. The gateway overwrites any payload sender with that actor.

3. An explicitly insecure local-development gateway may accept unsigned

payload identity. The constructor is documented as unsuitable for a network-facing deployment.

A valid native signature proves possession of the private key corresponding to the returned DID. It does not, by itself, decide which principal owns that DID, which tenant the principal belongs to, which scopes are permitted, or whether the requested capability is authorized. Those are separate trusted bindings and policy checks.

The gateway also applies its replay window independently from authentication. A correctly signed envelope can still be rejected as stale, too far in the future, or a replay of an already claimed message ID.

Trusted enrichment replaces caller claims

Identity resolution runs when an action carries identity hints or configured fleet routing needs identity for an unknown local capability. It also runs when the gateway marks a capability as identity-required or the capability credential policy requires a credential, issuer, or scope check. Delegated child actions cross the same resolution boundary as root actions.

The reviewed flow is:

1. The gateway authenticates the transport actor and collects the capability's

accepted credential issuers and required credential scopes.

2. It passes the actor and untrusted action identity to a

`TrustedIdentityResolver`.

3. The resolver returns the same actor plus optional `VerifiedTenant`,

`CredentialHandle`, and sanitized `IdentityContext` values from deployment-owned data.

4. The gateway rejects actor replacement, actor-kind changes, scope elevation,

expired membership or credentials, tenant mismatch, an unacceptable issuer, a missing required credential, and any mismatch between the sanitized identity and its verified tenant or credential.

5. The gateway stores only the validated values in `MessageContext`; the

runtime copies `context.resolved_identity` over `Action.identity` before preflight, queueing, persistence, and handler execution.

The default resolver is deny-all when no trusted identity provider is configured. The daemon's static directory is a controlled-deployment adapter: each entry is keyed by an authenticated principal and may contain verified tenant, opaque credential, sanitized identity, monotonic revision, revocation, and expiry state. Its loader rejects an empty directory, duplicate principals, zero revisions, revoked entries, expired entries, expired tenant membership, and expired credential handles.

An optional cache wrapper remains revocation-aware by asking the underlying resolver for its current revision before reuse. The provided default cache bounds are 10,000 entries and five seconds, further shortened by actor, tenant, or credential expiry. A resolver that cannot return a revision bypasses this cache rather than extending an identity decision without a revocation signal.

Tenant and credential authority stay deployment-owned

`VerifiedTenant` combines the canonical tenant reference with a membership ID, resolver-established roles and groups, verification time, and optional expiry. It is distinct from the tenant claim in `Action.identity`. Tenant-scoped execution and queries use the verified value.

`CredentialHandle` is an opaque runtime reference. It contains an identifier, issuer, scopes, optional tenant partition, and optional expiry, with the identifier redacted from its debug representation. The gateway checks

required scopes and accepted issuers. If the handle names a tenant, verified tenant membership must exist and match it.

The similarly shaped wire `CredentialRef` is a projection for protocol and audit context; it is not secret material and cannot be exchanged for authority without the trusted runtime handle. Capability contracts can require a credential and constrain issuers and scopes, but the deployment decides how a handle maps to a real provider credential.

`CredentialProvider` is the final secret boundary. It resolves a handle into short-lived `CredentialMaterial` for an immediate connector invocation. The provided material type redacts debug output and overwrites its owned byte buffer when dropped. Callers must still avoid copying those bytes into durable actions, errors, logs, receipts, traces, manifests, or connector route data.

Operational reads require ownership, tenant, and scope

Knowing an action, session, approval, transaction, receipt, audit, callback, resource, or event identifier does not authorize access to it. The reviewed query authorization service combines:

- a non-expired authenticated actor;
- object ownership or a trusted grant for the selected principal;
- a verified tenant and matching tenant selector when the actor is

tenant-bound;

- the exact object-specific read, write, or export scope;
- additional operation-specific scopes; and
- a separate sensitive-data scope before protected fields are disclosed.

Missing ownership, tenant, or scope information is denied. When the base read is authorized but the sensitive-data scope is absent, the service returns a field-disclosure decision that identifies fields to redact.

A `DelegatedAuthorityGrant` identifies one principal whose work is covered, the permitted scopes, an optional expiry, and an optional audit reason. The query service accepts it only from the authenticated principal, for the exact target principal, while it is unexpired, and for the exact requested scope or the wildcard `*`.

This is narrower than granting a global `object:read:any` scope. Delegation-chain continuity, hop limits, and remote delegation behavior are covered separately.

Approval authority is also resolved independently. A requester, action subject, and approver can be different principals; role or group text in an approval payload is not a substitute for trusted authority membership.

Connector fleets pin both principal and key identity

A connector fleet introduces two machine identities at every remote execution edge: the central gateway that signs the request and the connector-host replica that signs responses, callbacks, events, and lifecycle control messages.

For each replica, the registry stores the expected principal ID, principal kind, `peer_did`, trust domain, endpoint, immutable connector version, lease, and health revision. Route selection copies those values into a durable `RouteAssignment` together with the tenant, manifest digest, catalog and policy revisions, credential revision, quota policy, and fencing token.

The central dispatcher signs the outbound envelope as its bound gateway principal and verifies the response against the route's expected host principal and DID. The connector host trusts one configured gateway principal and gateway DID, verifies the signed request, and resolves that gateway to the host's fixed tenant, provider account, and opaque credential.

Connector event and stream-callback ingress performs the reverse check: signature DID, sender principal, trust domain, active replica lease, and durable route must agree.

Connector control-plane requests are signed by the host DID, while the host pins and verifies the control-plane DID in responses. A remote host receives a narrow lifecycle surface rather than registry-administrator or database credentials, and the replica identity is pre-provisioned before activation.

The connector manifest and the replica key have different lifecycles. The host requires the manifest's semantic agent identity to match its signing principal, but removes the replica DID from the published manifest. The trust domain stays part of the manifest identity; the per-replica DID remains in the registry. This permits horizontal replicas and key rotation without changing the immutable connector-version digest.

Credential revision is pinned separately from secret material

A tenant capability binding can name an opaque `credential_revision_ref`. Route assignment pins that reference before execution and carries it to the selected connector host. The host requires its credential handle and revision reference to be configured together and verifies the pinned revision against a deployment-owned `CredentialRevisionProvider` before dispatching the request to its gateway and connector.

The provided policy distinguishes:

Revision set	Meaning
`current_revision_ref`	Revision assigned to new work
`accepted_previous_revisions`	Older pinned work allowed during an explicit overlap window
`revoked_revisions`	Revisions denied immediately, even if an earlier route pinned them

The sets cannot overlap. Revocation wins, and an unavailable revision authority fails the request rather than guessing. The reference remains non-secret; it does not expose a token or prove that a provider accepted an operation.

Network and administrative surfaces stay bounded

Identity verification does not authorize an implementation to contact an arbitrary issuer or provider URL supplied by a caller. The reviewed OAuth introspection client uses a deployment-configured HTTPS endpoint, disables redirects, applies a five-second timeout, and limits the response body to one MiB. Its explicit HTTP exception accepts loopback destinations only for local development.

Connector-host control and message clients likewise use fixed deployment or registry destinations with transport, redirect, time, and body constraints. Exact values belong to their configuration and security reference pages rather than this identity model.

These trust paths do not expose general identity, registry, or credential administration. The remote connector host receives only registration, heartbeat, drain, and offline lifecycle operations; it does not receive a registry-administrator or direct database credential. Credential issuance, secret storage, OAuth consent, and provider-account administration remain outside the AIP wire surface.

Trust and data boundaries

- Payload `from`, `did`, `trust_domain`, delegated grants, tenant, account,

user, and credential references remain untrusted until the appropriate edge or resolver establishes them.

- Authentication proves an immediate actor under one issuer and audience. It

does not automatically grant capability, tenant, object, approval, or provider authority.

- Trusted enrichment may use payload values only as lookup hints. A mismatch is

an authorization failure, not a reason to copy the claim into runtime state.

- Tenant membership, downstream credentials, and approval memberships have independent expiry and revocation lifecycles.
- Raw provider secrets stay behind the credential-provider boundary. Opaque handles and revision references may be persisted; secret bytes may not.
- A DID key proves signature possession. Principal ownership, trust domain, active connector lease, durable route, and policy must also match.
- Each federation or delegation hop authenticates its immediate peer and applies local policy. A trust-domain label does not create transitive trust.
- Audit identity can be sensitive. Retention, redaction, and sensitive-field disclosure remain deployment policy even when an operation is authorized.

Design choices and trade-offs

Keeping `Principal` and `IdentityContext` on the wire preserves product and human context across transports and compatibility profiles. Treating them as claims requires an explicit resolver, but prevents a caller from selecting its own tenant, credential, or on-behalf-of authority.

Separating authentication from authorization lets a deployment use DID signatures, tokens, mTLS, webhooks, or authenticated adapters without changing the semantic action model. The cost is that every edge must clearly own its verification contract; merely selecting an `AuthScheme` enum is insufficient.

Opaque credential handles and revisions keep secret bytes out of durable AIP records and allow controlled overlap during rotation. They add a final online authorization check, and a revocation-service outage can deliberately block provider work.

Separating manifest identity from replica DID keeps connector artifacts stable across scaling and key rotation. The registry and route assignment must then retain exact per-replica identity, lease, and revision evidence.

Revision-aware caching improves resolver latency while preserving a fast revocation signal. Resolvers without that signal give up caching rather than silently extending trust.

What this model does not guarantee

- A well-formed `Principal` or `IdentityContext` is not authenticated.
- A valid signature is not authorization and does not establish tenant membership by itself.
- The authentication-scheme enum does not promise support for every scheme on every binding or deployment.
- A static identity directory is not a complete production IAM system.
- A credential reference or revision is not secret material and cannot prove that a provider credential is live.
- Memory zeroing of the owned credential buffer cannot erase copies made by other libraries, operating systems, or external providers.
- A trust domain does not imply universal or transitive federation.

- Scoped authorization does not prove connector qualification, provider correctness, or exactly-once external side effects.
- This page does not publish connectors beyond Cal.diy, Hermes Agent, Chatwoot, Dify, CrewAI, and Twenty.

Related pages

- Approvals and policy
- Delegation
- Profiles and connectors
- Security model
- Connector credentials and rotation