Security model

Use this page to evaluate the security boundaries in the reviewed AIP implementation. It identifies protected assets, attacker capabilities, deployment assumptions, fail-closed decisions, component-compromise scope, and risks that remain af

SECTION	Architecture and Security
TYPE	Architecture
PROTOCOL	AIP 1.0
SOURCE	docs/architecture/security-model.md

The page describes version `1.0.0` of the Rust workspace at source revision `97be86e9efedf07ecf1783b03800f683f107fb04`. It is a source-backed model, not a penetration test, production certification, or assessment of one deployed environment.

Security objective

The implementation aims to ensure that only an authenticated and authorized principal can cause an action or read protected lifecycle state. The objective also requires tenant, connector, credential, and provider-account selection to come from trusted deployment state. Every external effect must remain correlated with durable intent and evidence.

Five properties support that objective:

Property	Required security result
Authenticity and integrity	A receiving boundary verifies the immediate actor and rejects modified signed content
Authorization and isolation	Capability, object, tenant, approval, route, and credential decisions use server-owned state
Secret minimization	Raw credentials stay at the narrow invocation boundary and out of protocol records
Replay and concurrency safety	Repeated messages, action intents, workers, replicas, and control operations are fenced at their own layer
Audit and recovery	Durable records identify what was accepted, routed, attempted, decided, and left uncertain

These properties do not make every envelope confidential. Native signatures protect integrity and signer possession, while transport security or an explicit session cipher must provide confidentiality where required.

Protect the assets that carry authority

Asset	Security significance
Signing and callback-encryption keys	Establish native peer identity or protect persisted Agent2Agent (A2A) push credentials
Bearer tokens and provider credentials	Grant edge or external-product authority
Trusted identity, tenant, and approval records	Convert caller claims into deployer-authorized context
Capability and query policy	Decide which actor may execute or inspect which object
Idempotency, replay, queue, and transaction state	Fence repeated intent and preserve uncertain work across failures
Connector catalog, route assignments, leases, and credential revisions	Select one tenant-owned provider account and one admitted host
Release packages and independent evidence	Decide which immutable connector artifact may enter the fleet
Results, events, receipts, and audit records	Preserve operational, customer, and security-sensitive evidence
Availability budgets and storage	Prevent one workload from silently consuming every execution or recovery path

Identifiers, digests, opaque credential handles, and revision references are not secret material by themselves. They still require integrity and access control because changing one can redirect authority or corrupt evidence.

State the assumptions before evaluating controls

The source assumes that a deployment:

- provisions the correct trusted decentralized identifier (DID), token issuer,

tenant, approval, release, and evidence-signer roots;

- protects private keys, bearer tokens, database credentials, provider secrets,

and callback-encryption keys outside AIP payloads;

- isolates the central daemon, registry administration, connector hosts, and

their operating-system identities according to the intended topology;

- supplies a durable backend with the advertised single-process or shared

coordination guarantees;

- terminates and authenticates network transports according to the selected

binding, including TLS and any proxy boundary;

- maintains sufficiently accurate time for expiry, skew, lease, and replay

decisions;

- controls DNS, certificates, network policy, and deployment configuration

around allowed remote destinations; and

- treats connector and provider responses as untrusted data until their schema,

route, and product-specific checks pass.

The implementation validates many inputs to these assumptions, but it cannot protect a private key disclosed by the secret store. It also cannot protect against a database administrator that can rewrite all state or a process running with unrestricted host access.

Model an active untrusted caller

The model assumes an attacker may:

- create arbitrary well-formed or malformed protocol payloads;
- assert another principal, tenant, credential, account, callback, or

delegation in caller-controlled fields;

- replay, reorder, delay, duplicate, or tamper with traffic;
- guess object identifiers and query protected lifecycle endpoints;
- disconnect or cause ambiguous responses around an external side effect;
- send forged or repeated provider webhooks;
- request callback or connector destinations that target internal networks;
- induce concurrent workers, stale leases, retries, or old control messages;

and

- publish an artifact or evidence document that should not pass admission.

A valid credential or signed peer can also be malicious within its granted authority. Authorization, tenant isolation, route binding, limits, and audit therefore remain necessary after authentication.

Cross explicit trust boundaries

```
MERMAID
flowchart LR
    U["Untrusted client or foreign protocol"] --> E["Authenticated edge"]
    E --> G["Gateway trust and policy"]
    G --> R["Runtime lifecycle"]
    R --> C["Pinned connector route"]
    C --> H["Isolated connector host"]
    H --> P["External product"]
    H -->|signed events and callbacks| G
    A["Release and evidence authorities"] -.-> M["Admission boundary"]
    M -.-> C
    S["Deployment secret and identity systems"] -.-> E
    S -.-> G
    S -.-> H
```

The text equivalent is:

1. A native or compatibility edge authenticates the immediate caller before it

invokes a trusted gateway entrypoint.

2. The gateway binds that actor to trusted identity and tenant state, claims

replay protection, and applies operational authorization.

3. The runtime governs capability execution, approvals, idempotency,

transactions, lifecycle persistence, and recovery.

4. Fleet work uses a durable route that pins one admitted connector version,

tenant instance, replica, peer identity, lease, and credential revision.

5. The connector host verifies that route and central peer before resolving a

provider credential inside its process.

6. Provider data, stream chunks, callbacks, and events return through their own

signature, route, replay, and schema checks.

7. Separately trusted release and evidence authorities admit immutable

connector state; deployment systems supply identities and secrets without becoming payload fields.

Trust is local to each arrow. A signature, token, manifest, trust-domain label, route, or approval at one boundary does not automatically authorize the next.

Bind native ingress to an expected principal

The default gateway requires a signed native envelope, a sender, and replay protection. It canonicalizes the complete envelope and removes only `security.signature` from the signed representation. An application field also named `signature` remains covered.

Ed25519 verification returns the `did:key` that possessed the signing key. The gateway then performs the separate trust decision: that DID must be explicitly bound to a server-owned principal, and `Envelope.from` must match its ID and kind. Key possession alone cannot choose a principal or tenant.

An already authenticated transport can instead call the matching trusted entrypoint. The gateway replaces the payload sender with the actor supplied by that edge. This is safe only when the caller of that in-process API is itself a trusted authentication boundary.

The daemon's native bearer comparison is constant-time and binds the accepted token to a configured principal. Unsigned native traffic without that authenticated actor requires explicit insecure-development mode, which is restricted to a loopback listener. A non-loopback configuration requires a public HTTPS origin rather than treating plain public HTTP as secure.

The authentication-scheme enum also names OAuth2, mTLS, API-key, and webhook forms. Its existence does not prove that every listener implements every scheme. Review the selected edge, not the enum, for actual verification.

Separate authentication, authorization, and identity enrichment

Payload `Principal`, `Envelope.from`, `Action.identity`, tenant, credential, delegation, and trust-domain values are claims until a trusted boundary establishes them.

The gateway's identity resolver receives the authenticated actor and uses caller values only as lookup hints. It rejects actor replacement, scope elevation, expired authentication or membership, tenant mismatch, an unacceptable credential issuer, missing required scopes, and conflicts between resolved identity and the verified tenant or credential handle. The runtime copies only resolved identity over the action before persistence or execution.

Authorization remains layered:

- capability policy checks principal kind, scope, delegation depth, risk, and

human-approval requirements;

- approval authority verifies the approver independently of payload role text;
- query policy checks authentication, object ownership or delegated authority,

tenant selection, operation-specific scope, and sensitive-field disclosure; and

- connector-host approval evidence is bound to the exact approval, action,

capability, decision, status, and tenant.

Knowing an action, session, approval, transaction, receipt, or callback ID does not grant read access. Authentication also does not grant provider-account selection.

The detailed identity objects and resolver flow are owned by Identity and trust.

Fence replay, repeated intent, and stale workers separately

One mechanism cannot solve every duplicate-delivery problem.

Mechanism	Rejects or fences	Important limit
Gateway replay claim	Reuse of a native message ID inside its accepted timestamp window	A new message ID can still describe the same action intent
Webhook hash-based message authentication code (HMAC) and skew check	Forged payloads and deliveries outside the accepted time window	The receiving connector must also deduplicate the delivery ID
Runtime idempotency reservation	Concurrent or repeated action intent bound to actor, capability, context, and input	It cannot make a non-idempotent provider atomic
Queue and callback leases	Stale workers completing state owned by another attempt	Backend coordination must match the deployment's process count
Route and replica fencing	A different or stale host acting for the pinned action	It does not determine provider truth after an ambiguous response
Admission and lifecycle journals	Conflicting package revisions or stale host-control transitions	Administrator compromise remains outside application-level fencing

The gateway checks message time and claims replay state before action dispatch. The shared store must make that claim atomic across gateway instances.

For external mutations, the runtime can persist intent and provider-operation checkpoints, while the connector can use a provider idempotency facility or reconciliation lookup. The code does not promise exactly-once provider effects.

Keep secrets behind opaque handles

Trusted runtime context carries an opaque `CredentialHandle`, not token bytes. The handle records an identifier, issuer, scopes, tenant partition, and expiry. A deployment-owned credential provider resolves it into short-lived `CredentialMaterial` only at connector invocation.

The material's debug output is redacted and its owned buffer is overwritten on drop. Bearer-token wrappers follow the same redaction and clearing pattern. This reduces accidental disclosure, but cannot erase copies made by HTTP libraries, operating systems, crash dumps, or external providers.

`ExecutionContext` is intentionally not serializable. Its redaction policy denies common credential pointers and can mark all input or output sensitive. Connectors and observability code must still avoid placing raw payloads, authorization headers, secrets, or unbounded provider errors into logs, events, receipts, metrics, or stored actions.

For fleet work, the route contains an opaque credential revision, never the secret. The host requires its handle and revision configuration to agree and consults deployment-owned revision policy. Revocation fails the request even when an older action pinned that revision; an unavailable revision authority fails closed.

Constrain a connector host to one pinned route

Central route resolution uses verified tenant and capability context, not a caller-selected endpoint or provider account. The durable assignment pins the action, capability, tenant, instance, replica, immutable version, manifest digest, catalog and policy revisions, credential revision, quota reference, peer principal and DID, transport profile, capacity reservation, and fence.

The central dispatcher signs the action for the assigned host and validates the signed response against the expected peer. The host independently verifies:

- the native signature and configured central gateway DID and principal;
- recipient and sender identity;
- every route coordinate against its fixed instance and version;
- active lease, health sequence, non-draining state, and local capacity;
- verified tenant and provider-account mapping;

- credential revision and any supplied approval authorization; and
- capability, mode, schema, and admitted handler support before provider work.

Responses are signed by the host. Stream callbacks and normalized provider events traverse the reverse boundary: the central ingress checks the host DID, principal, trust domain, active replica, manifest or durable action route, and replay state before accepting them.

A compromised host is intended to be contained to its configured connector instance, immutable version, provider credential, and signed host identity. It does not receive registry-administrator credentials and cannot self-admit a new release. Operating-system or network isolation must enforce that intended containment.

Require independent release evidence

Connector admission is the only supported bridge from build evidence to mutable fleet catalog administration. A release authority signs one complete package that binds the immutable artifact digest, manifest, implementation claims, topology, instances, replicas, bindings, and evidence set.

Seven evidence families are mandatory: artifact signature, software bill of materials, provenance, conformance, vulnerability policy, license policy, and revocation observation. Each has a policy-specific trust-root set, exact artifact and manifest subjects, document digest, signed outcome, issue time, and hard expiry. The default policy also requires the package signer and all evidence signers to be distinct.

Admission derives registry attestation state after verification; a release payload cannot self-assert that its scans passed. Package size, evidence size, topology counts, owner, clocks, versions, schema, and implementation support are bounded before catalog mutation.

This protects against artifact substitution and one broad release key asserting every policy result. It does not prove that a trusted evaluator is correct or that an admitted connector is harmless. Release and evidence root compromise, malicious source accepted by every gate, and deployment-time image replacement remain supply-chain risks outside a signature check alone.

Restrict outbound destinations

Callback and native remote clients apply destination policy before sending trusted data. The reviewed HTTP callback policy starts with no allowed hosts, disables plaintext HTTP, rejects private or non-public addresses, disables redirects, limits time and response size, and forbids URL user information and fragments.

DNS is resolved under policy, and HTTP requests use the selected pinned address. Private networks and plaintext HTTP require explicit controlled-deployment exceptions.

External native callbacks require an Ed25519 signer. A2A push credentials can be persisted only as AES-256-GCM ciphertext when the deployment supplies the shared encryption key required by every recovering replica. Callback payload signing, destination authorization, credential encryption, and delivery state are separate controls.

Remote connector HTTP policy can derive the host allowlist from the already pinned registry endpoint while retaining scheme, address-range, timeout, body, signature, and expected-peer checks. An endpoint present in caller payload does not become an allowlist entry.

These controls reduce server-side request forgery and peer substitution. They do not replace egress firewalls, service-mesh policy, certificate operations, DNS security, or segmentation.

Preserve durable coordination and bounded work

Replay, idempotency, queue, callback, approval, transaction, and route safety depend on atomic backend operations. In-memory state is process-local. The local file runtime is durable for one process, while the PostgreSQL runtime bundle advertises shared coordination and uses transactional leases and compare-and-set transitions.

Runtime callback and reconciliation work have independent concurrency budgets. Remote action scheduling has global and tenant bounds, and interfaces impose bounded timestamps, token sizes, response bodies, package documents, queue attempts, and retry windows at their respective boundaries.

Bounds reduce accidental or adversarial resource amplification. They are not a complete denial-of-service defense. Network admission control, rate limiting, database capacity, tenant quotas, process limits, and upstream protection remain deployment responsibilities.

Interpret audit evidence accurately

The runtime records lifecycle events, results, approval and transaction state, callback attempts, receipt chains, and recovery reports. Fleet state retains route assignments, leases, settlements, admission operations, and signed release evidence.

Receipt hashes and canonical digests can reveal changes within retained data. They do not prevent an administrator from deleting an entire store, provide an external timestamp, or establish non-repudiation without an independently protected signing and retention system.

Audit access is itself authorized and may require a sensitive-field scope. Retention and redaction policy must balance investigation needs with customer data and secret minimization.

Review component-compromise scope

Compromised component	Intended containment	Security consequence that remains
Untrusted client	Cannot establish a different signed or transport-bound actor	Can act within legitimately granted scopes and consume allowed capacity
Trusted profile or authentication edge	Still faces gateway policy, tenant, runtime, and route checks	Can impersonate the actors that the gateway trusts it to establish
Central daemon or its signing key	Product secrets remain in connector hosts and routes stay auditable	Can authorize, route, inspect central state, and send trusted host requests
Registry reader path	Hosts still require signed central requests and exact fixed identity	Corrupted route state can misdirect central selection if the daemon accepts it
Registry administrator or database	Host and release signatures provide partial independent evidence	Catalog, route, lease, or journal integrity can be subverted
One connector host	No catalog administration and one intended instance or credential boundary	Can misuse its provider credential and emit messages as its trusted host identity
Release package signer	Cannot alone create all distinct evidence outcomes	Can nominate malicious content and topology for the other authorities to evaluate
One evidence authority	Other evidence kinds and package signature remain required	Can falsely approve the evidence family whose root it controls
Provider account or API	AIP retains intent, route, and reconciliation evidence	Provider can return malicious data or commit effects contrary to expectations
Complete deployment host or secret store	Application-level boundaries offer limited protection	Keys, processes, memory, databases, and provider credentials may all be exposed

This table describes intended blast radius under the documented topology. It does not claim process isolation against software with unrestricted host or database administrator access.

Residual-risk register

Residual risk	Required deployment or design response
Key or token compromise	Revoke and rotate the exact trust binding; investigate affected signed or authenticated work
Incorrect identity, tenant, or approval authority	Treat resolver and authority configuration as security-critical reviewed code and data
Database administrator compromise	Use database access control, encryption, backups, immutable external audit, and change monitoring
Ambiguous provider mutation	Use provider idempotency and reconciliation; do not retry blindly or silently reroute
Malicious connector or dependency	Preserve isolated hosts, least-privilege provider credentials, independent admission evidence, and egress policy
Sensitive data in payloads or logs	Apply schema minimization, redaction, access control, retention, and incident review
Resource exhaustion	Add edge rate limits, tenant quotas, storage capacity controls, and supervisor limits
DNS, TLS, proxy, or network misconfiguration	Enforce deployment network policy and validate the effective route outside application source
Lost or incomplete audit history	Export to independently protected retention where the threat model requires it
Unsupported authentication assumption	Verify the concrete transport edge instead of relying on the `AuthScheme` vocabulary

Security-review checklist

Before approving a deployment or change, answer:

1. Which boundary authenticates the immediate caller, and which trusted gateway entrypoint does it use?
2. Which principal, tenant, scope, approval, and credential authority records are trusted, revocable, and time-bounded?
3. Which store durability class backs replay, idempotency, queue, callback, transaction, and route state for the actual replica count?
4. Which exact connector instance, peer DID, artifact digest, credential revision, and lease can receive the action?
5. Which provider operations are idempotent or reconcilable after an ambiguous response?
6. Which callback and connector destinations are allowlisted, address-checked, TLS-protected, signed, and size-bounded?
7. Where can raw credentials or sensitive payloads enter memory, logs, traces, events, receipts, backups, or crash artifacts?
8. Which release and evidence roots can admit an artifact, and how are they rotated or revoked independently?
9. What is the blast radius if the client edge, daemon, registry, one host, one evidence signer, database, or secret store is compromised?

10. Which claims have been verified in the deployed environment rather than inferred from source support?

What this model does not guarantee

- Signatures do not provide confidentiality, authorization, tenant membership, key custody, or transitive trust.
- A trust-domain label, manifest, ready process, or admitted artifact does not authorize an action.
- A static identity directory is not a universal IAM system.
- Memory clearing cannot erase secret copies outside the owned buffer.
- Leases, idempotency, and durable intent do not guarantee exactly-once external effects.
- Hash chains do not prevent record deletion or replace independently protected audit storage.
- Source support does not prove secure configuration, production enablement, conformance, availability, or connector qualification.
- This page does not assess any connector beyond Cal.diy, Hermes Agent, Chatwoot, Dify, CrewAI, and Twenty.

Related pages

- Identity and trust
- Gateway
- Runtime
- Connector admission
- Connector credentials and rotation