SADAR™ Semantic Registry Architecture

Registry Core Security

Version 0.4.1 — June 2026

Overview

This document specifies the security architecture for the SADAR™ Semantic Registry. It covers three distinct authentication and authorization contexts:

- Registry access controls — how registry operators constrain replication and forwarding access between registries (Section 1)
- Registry-to-registry security — how registries authenticate to one another for replication and forwarding, using the Registry of Registries as trust anchor (Section 2)
- Requester authentication — how agents and other entities authenticate to the local registry to issue discovery queries (Section 3)
- Service invocation authentication — how a requesting agent establishes trust with a selected service agent at runtime, after discovery is complete (Section 4)

Sections 1–3 govern interactions in which the registry is an active participant. Section 4 governs agent-to-agent interactions at invocation time, in which the registry is explicitly not a participant — its role ends at discovery.

1. Pull Request Controls and Charging

A providing registry has full authority to constrain replication access. Just as a service provider may enforce usage limits and access controls on consuming entities, a registry operator may enforce equivalent controls on registries requesting to pull its content. These controls are an operational and commercial accountability of the providing registry administrator, not a protocol-level negotiation.

1.1 Access Control Mechanisms

The following mechanisms may be applied independently or in combination. All mechanisms are enforced by the providing registry before any content is returned.

Control Mechanism	Description	Operational Notes
Rate Limits	Cap the number of pull requests per time window (e.g. requests/minute, requests/day)	Prevents a single subscribing registry from saturating the providing registry's bandwidth. May be differentiated by subscriber tier or commercial agreement.
Payload Size Restrictions	Cap the maximum response payload size per pull request (record count, byte size, or both)	Protects providing registry resources and forces subscribing registries to paginate large pulls. Prevents bulk data exfiltration in a single call.
Allow Lists	Explicit enumeration of registry URNs permitted to request replication from this registry	Positive-control model: only registries explicitly authorized may pull. Required for internal proprietary or commercially restricted registries. Evaluated before any other control.

Block Lists	Explicit enumeration of registry URNs denied replication access, regardless of other criteria	Negative-control model: specific registries are excluded even if they would otherwise satisfy allow-list or authentication criteria. Used to enforce commercial exclusions, competitor restrictions, or revoke previously granted access.
Authentication Requirement	Replication requests must present a valid registry identity token	Ensures only legitimately registered registries may pull. Unauthenticated pull requests are rejected regardless of allow/block list configuration.
Scope Restriction	Limits replication to a declared subset of the registry's content (e.g. specific NAICS branches, providers, manifest versions)	A registry may publish its full catalog for forwarding-mode queries while allowing replication of only a defined subset. Commercial tiers may expose different replication scopes at different price points.

Allow lists and block lists are evaluated first, before authentication. A registry on the block list is rejected regardless of whether it holds a valid identity token. A registry not on the allow list (when an allow list is configured) is rejected regardless of authentication status. This ordering prevents authentication infrastructure from becoming a bypass vector for access control.

1.2 Charging for Forwarding and Replication

Registry operators may charge rates to calling registries for forwarding and replication services. The charging model follows exactly the same settlement patterns defined in the CogniWeave™ Payment Architecture Specification — the same models, the same settlement providers, the same mechanics, applied at the registry-to-registry level rather than the entity-to-service level.

- Advertising: The providing registry's manifest (held in the Registry of Registries) declares the supported settlement models, pricing tiers, and settlement provider for replication and forwarding access, using the same paymentSupport structure used by service manifests.
- Negotiation: At connection establishment between two registries, the providing registry returns settlement configuration — billing model, cost structure, settlement recipient identifier — in the same structure as a service connection establishment response.
- Settlement: Financial settlement proceeds through the designated settlement provider (Stripe, PayPal, X.402 network, etc.) without the providing registry or the standards body ever handling payment instruments directly.
- Enforcement: Access to replication or forwarding is governed by the requesting registry's account standing with the providing registry, not by per-request payment authorization. This is consistent with the SADAR philosophy of avoiding runtime payment dependencies in the execution path.

The providing registry may configure different commercial tiers that expose different replication scopes, rate limits, or payload allowances. The manifest advertises available tiers; the requesting registry selects a tier at connection establishment.

2. Registry Replication and Forwarding Security

The security model for registry-to-registry interactions is a direct extension of the entity-to-service security model. The same principles apply: signed manifests, OIDC-based identity, JWE-encrypted usage keys, mutual TLS, and registry isolation. What changes is the level of abstraction: the Registry of Registries (RoR) plays the same trust-anchor role for registries that a local registry plays for entities.

> *The full treatment of replication protocol mechanics, forwarding controls, charging, and multi-hop identity is documented in SADAR™ Registry Replication Security. This section describes the security architecture; that document describes the operational protocol.*

2.1 The Registry of Registries as Identity Bridge

The standards-body-operated Registry of Registries is not merely a discovery directory for registry descriptors. It is the trust anchor for the entire registry-to-registry identity layer. The RoR assigns registry identities, holds signed registry manifests, and issues the tokens that allow registries to authenticate to one another — exactly as a local registry assigns entity identities, holds signed service manifests, and provides the discovery information entities need to authenticate to services.

This symmetry is intentional. The same tooling, the same cryptographic primitives, and the same operational patterns that govern entity-to-service interactions govern registry-to-registry interactions. The only difference is the scope at which they operate.

Dimension	Entity ↔ Registry (existing model)	Registry ↔ Registry (extended model)
Trust anchor	The Semantic Registry acts as trust anchor for entities (agents, services) within its scope. It assigns entity IDs, holds signed manifests, and is the authoritative source for entity discovery.	The Registry of Registries acts as trust anchor for registry operators across the ecosystem. It assigns registry IDs, holds signed registry manifests, and is the authoritative source for registry discovery.
Identity issuance	An entity registers with its local registry and receives a registry-issued entity URN.	A registry operator registers with the standards body RoR and receives a RoR-issued registry URN.
Manifest signing	Entities sign their own service manifests (JWS/ES256) prior to upload. The registry stores manifest hash + signature; it never holds the entity's private key.	Registry operators sign their own registry manifests (JWS/ES256) prior to upload to the RoR. The RoR stores manifest hash + signature; it never holds the registry's private key.
Token assertion at runtime	At service invocation, the client entity asserts a signed identity token to the provider service. The provider validates the token against the registry's JWKS.	At replication or forwarding time, the requesting registry asserts a signed identity token issued against the RoR. The providing registry validates the token against the RoR's JWKS.
License and payment	Service manifests advertise supported settlement models and licensing terms. Usage keys and settlement metadata are exchanged	Registry manifests advertise supported replication models and licensing terms for access. License keys and settlement metadata for

	at connection establishment per the Payment Architecture Specification.	replication fees are exchanged at pull/forwarding establishment using identical mechanisms.
Right of refusal	A provider service may refuse a connection from a client entity whose manifest characteristics are incompatible (compliance posture, governance framework, entity type).	A providing registry may refuse a replication or forwarding request from a requesting registry whose registry manifest characteristics are incompatible, or which does not appear on the providing registry's allow list.

The RoR's blast radius is bounded identically to a local registry's blast radius: it is limited to identity issuance and manifest storage. It is not a runtime participant in replication or forwarding data flows. Once two registries have authenticated to one another, all subsequent replication and forwarding interactions occur directly between them — the RoR is not in the data path.

2.2 Registry Operator Registration

Before a registry instance may participate in inter-registry replication or forwarding, its operator must register with the standards body through the RoR. This process mirrors entity registration with a local registry:

- Operator authentication: The registry operator authenticates to the RoR using the organization's credentials (mTLS + OIDC Client Credentials flow).
- Registry ID assignment: The RoR assigns a globally unique registry URN to the registry instance (e.g. urn:sadar:registry:acme-corp:primary).
- Registry manifest publication: The operator prepares a registry manifest describing the registry instance's scope, NFRs, supported replication models, licensing terms, and settlement configuration. The manifest is signed locally by the operator's private key (JWS/ES256) and uploaded to the RoR. The RoR stores the manifest hash and signature; it never holds the operator's private key.
- OIDC endpoint declaration: The operator's registry OIDC endpoint URI is embedded in the signed registry manifest, meaning the endpoint address is covered by the operator's signature and cannot be redirected undetected.

2.3 Registry-to-Registry Authentication

When a registry wishes to initiate a replication pull or forward a query to another registry, it must authenticate using a token issued by the RoR. The providing registry validates that token independently against the RoR's JWKS — the RoR is not involved in this validation at runtime.

2.3.1 Authentication Flow

- The requesting registry authenticates to the RoR's OIDC endpoint (Client Credentials flow, mTLS).
- The RoR issues a signed JWT asserting the requesting registry's identity and granted scopes, targeted to the specific providing registry (audience claim = providing registry URN).

- The requesting registry presents this token to the providing registry's replication or forwarding endpoint.
- The providing registry fetches the RoR's JWKS (or uses a cached version within its TTL) and validates the token signature, expiry, and audience claim locally. The RoR is not contacted at this step.
- The providing registry checks the requesting registry URN against its allow list and block list.
- If all checks pass, the providing registry returns a JWE-encrypted usage key for this replication or forwarding session — encrypted to the requesting registry's public key (same pattern as entity usage key delivery).
- Subsequent pull requests in this session present the usage key as the session credential. The usage key has its own TTL independent of the RoR token.

2.3.2 Registry-to-Registry Access Token Claims

Claim	Type	Description	Notes
iss	Standard (OIDC)	Issuer — the RoR's OIDC endpoint URI	Fixed per RoR instance
sub	Standard (OIDC)	Subject — the requesting registry's URN	Assigned at registry registration
aud	Standard (OIDC)	Audience — the providing registry's URN	Targeted; token is not reusable across registries
jti	Standard (OIDC)	Unique token identifier / request nonce	Prevents replay
iat / exp	Standard (OIDC)	Issued-at and expiration timestamps	Short-lived; TTL defined by RoR policy
scope	Standard (OAuth)	Granted operation scopes (e.g. sadar:registry:replicate, sadar:registry:forward)	Scope set at RoR registration or by commercial agreement
cw_registry_id	CogniWeave ext.	Requesting registry's URN	Matches sub; included for explicit registry identification
cw_manifest_iri	CogniWeave ext.	IRI of the requesting registry's current active manifest held in the RoR	Providing registry can fetch and validate the requesting registry's manifest
cw_usage_key	CogniWeave ext.	Usage key for this replication/forwarding session, encrypted as JWE to the providing registry's public key	Same JWE pattern as entity-level usage keys

2.4 Multi-Hop Forwarding and Identity

When a query propagates across multiple forwarding hops, the identity and trust model must remain coherent across the chain. Two principles govern this:

- Each hop authenticates independently: Each registry in the forwarding chain presents its own RoR-issued token to the next-hop registry. A forwarded request does not carry the originating registry's token through the entire chain — each hop is a distinct authenticated registry-to-registry interaction. This prevents a compromised intermediate registry from forwarding with a token it did not earn.
- Originating entity scope propagates with the query: The scope asserted by the originating entity's manifest is included in the forwarding request payload (not in the registry-to-registry authentication token). Intermediate and terminal registries may use this scope to enforce their own access policies for the entity's query, independent of the inter-registry authentication.

A providing registry receiving a forwarded query validates two independent identity assertions: (1) the forwarding registry's RoR-issued token, establishing that it is a legitimately registered registry; and (2) the originating entity's manifest scope claim, establishing that the query was initiated by an entity with an active, in-scope manifest. Either assertion may independently result in the query being refused.

2.5 Internal Proprietary Registry Exclusion

An internal proprietary registry does not register with the RoR and does not participate in the registry-to-registry authentication protocol. It has no registry URN in the RoR namespace, no RoR-held manifest, and no OIDC endpoint for inter-registry authentication. It cannot be a replication source or a forwarding target for any registry outside the enterprise boundary.

This exclusion is categorical and enforced at the network and protocol boundary, not merely by configuration. An internal proprietary registry that is unreachable from outside the enterprise network satisfies this requirement architecturally. Additional enforcement through the registry's access control layer is defense-in-depth.

2.6 Security Accountability Summary

Concern	Owner	Mechanism / Notes
Registry private key	Registry operator	Never leaves the registry operator's environment. The RoR never holds registry private keys.
Registry manifest integrity	Registry operator + RoR	JWS signature + hash storage in RoR. Any post-upload tampering is detectable by any validating party.
Registry identity token issuance	RoR OIDC endpoint	Tokens issued by the RoR on authenticated registry-to-registry interactions.
Token validation (R2R)	Providing registry	JWKS fetch from RoR + local signature verification. RoR not involved at runtime.
Usage key generation (R2R)	Providing registry	Generated at replication/forwarding session establishment; JWE-encrypted to requesting registry's public key.

Replication license enforcement	Providing registry	Providing registry validates usage key on each pull request; rejects requests with expired or invalid keys.
Allow/block list enforcement	Providing registry	Applied before token validation. A blocked registry cannot proceed even with a valid token.
Rate and payload limit enforcement	Providing registry	Applied per authenticated requesting registry URN. Limits may vary by commercial tier.
Requester entity registration	Local registry	Entity must be registered and hold an active manifest before issuing queries.
Requester manifest scope enforcement	Local registry	Registry validates query criteria against entity's manifest scope at every query.
RoR blast radius	RoR architecture	Limited to registry discovery and identity issuance. RoR is not a runtime participant in replication or forwarding data flows.

3. Requester Authentication and Authorization

Every entity that issues a query to a registry — whether a direct consumer agent, an orchestrator, or a pipeline component — must be a registered entity of that registry with an active, valid manifest. Anonymous or unauthenticated query access is not permitted.

This requirement serves two governance functions. First, it creates an auditable record of who queried the registry and under what manifest terms, supporting compliance reporting and forensic investigation. Second, it enables the registry to enforce manifest-scope constraints on query criteria — an entity's registered manifest defines the scope of what it is permitted to search for, not merely what it is permitted to invoke.

3.1 Registration Prerequisite

Before an entity may issue queries:

- The entity must be registered in the local registry and assigned an entity URN.
- The entity must have an active, signed manifest on file with the registry. The manifest declares the entity's identity, its operational scope, its compliance posture, and any governance constraints it operates under.
- The entity's manifest must be in the Active lifecycle state. An entity whose manifest is deprecated, past its deprecated_date, or superseded may not issue queries until it publishes and activates a replacement manifest.

3.2 Query Session Authentication

The authentication flow for a querying entity mirrors the client-side flow in the entity-to-service security model:

- The entity authenticates to the local registry's OIDC endpoint using its private key (Client Credentials flow, mTLS).
- The registry validates the entity's credentials and confirms its registration status and manifest currency.
- The registry issues a signed query session token scoped to the entity's manifest-declared query permissions. This token carries the Entity URN, Agent ID, and granted operation scope as standard claims, along with a jti (JWT ID) and iat/exp timestamps. The jti + exp combination prevents replay: each token is unique, short-lived, and targeted to this registry instance via the aud claim.
- The entity presents this token with each query request.
- The registry validates the token and enforces scope constraints against the query criteria before processing.

3.3 Authentication TTL and Discovery TTL

There are two independent TTL concepts in discovery that must not be conflated.

The Authentication TTL defines the period for which the requesting agent's identity has been authenticated. Once expired, it must reauthenticate to access the registry. Because Client Credentialing is a non-interactive, machine-to-machine flow, token refresh is a silent, automatic

operation: the agent detects expiry (proactively by inspecting exp, or reactively on a 401 response), re-presents its private key credentials to the registry's OIDC endpoint over mTLS, and receives a new signed JWT with a fresh jti, iat, and exp. Re-authentication does not restart discovery.

The Discovery TTL is the period for which the discovery results are valid. It is governed by the selected serving entry's manifest-declared TTL — the serving entry, not the requester or the registry, defines the maximum lifetime of a cached candidate set. When it expires, the cached result is invalidated, the next discovery request triggers a full re-execution of the search, and the requester must evaluate and select from the new candidate set. The expired candidate list is retained for audit purposes; it is not used for routing after expiry.

3.4 In-Flight Request Behavior at Authentication Expiry

The governing principle is commit-or-reject at admission, not mid-execution. When the registry accepts a discovery request, it validates the authentication token at that moment. If the token is valid at admission, the request executes to completion regardless of whether the token expires during processing. The registry does not re-validate mid-execution, avoiding a race condition where a long-running forwarding chain could span a token boundary and produce a partial result with no defined compensating transaction.

- Token valid at admission → request completes. The registry commits to returning the full result, including any forwarding sub-queries, under the authentication that was valid when the request entered.
- Token expired before admission → request rejected. The registry returns a 401 before any processing begins. The agent re-authenticates and resubmits.
- Token expires during a forwarding chain: Because the forwarding request was issued under a valid session, intermediate registries continue processing. The originating registry assembles and returns the complete aggregated result. The agent's next request will trigger re-authentication.

Implementers note: a very short Authentication TTL combined with a high-latency forwarding chain could result in the agent needing to re-authenticate before it can act on a result still being assembled. The searchAndInvoke reference implementation handles this by proactively refreshing the authentication token when exp is within a configurable threshold — defaulting to 30 seconds — so that the token presented on the next request is already fresh before the prior result is returned.

3.5 Manifest Scope Enforcement

The registry enforces that the entity's query criteria fall within the scope declared in its active manifest. An entity registered with a manifest scoped to NAICS 62 (healthcare) and APQC PCF node 8 may not issue queries targeting NAICS 52 (finance) or PCF node 3 — these are outside its declared scope.

Scope violations are returned as explicit errors, not silently narrowed results. The entity's query is rejected with a scope-violation response identifying which criteria exceeded the entity's manifest scope. Silently narrowing a query would produce a result set the consumer could mistake for complete.

3.6 Forwarded Query Scope Propagation

When the local registry forwards a query outward, it includes the originating entity's manifest scope in the forwarding request payload. This enables intermediate and terminal registries to enforce their own access policies with respect to the originating entity, independent of the inter-registry authentication.

A providing registry receiving a forwarded query may accept the query as presented, narrow the query to the intersection of the forwarded scope and its own allow policy, or reject the query if the originating entity's manifest characteristics are incompatible with its governance requirements.

3.7 The searchAndInvoke Tool

In practice, the searchAndInvoke tool is provisioned to the requesting agent through the agent's native tooling mechanism — MCP, A2A, or a framework-specific means. This tool encapsulates the full discovery lifecycle: authentication, token refresh, request construction, result caching, TTL tracking, and selection acknowledgment. The calling agent interacts with a single search interface without managing any of these mechanics directly.

A reference implementation of this tool with MCP bindings is published as part of the SADAR open-source reference implementation.

3.8 Accountability Table

Control	Owner	Description
Requester entity registration	Requesting entity (agent/orchestrator)	Entity must be registered in the local registry and assigned an entity URN before it may issue queries.
Requester manifest validity	Requesting entity + registry	The entity's manifest must be active (not deprecated, not superseded) at the time of the query.
Query token issuance	Local registry	The registry issues a signed query session token to the requesting entity on authentication. This token scopes the entity's permitted query operations.
Manifest scope enforcement	Local registry	The registry enforces that query criteria fall within the entity's registered manifest scope. Queries outside manifest scope are rejected with a scope-violation error, not silently narrowed.
Forwarding token propagation	Local registry	When the registry forwards a query outward, the forwarding token includes the scope asserted by the originating entity's manifest. The providing registry may use this to enforce its own access controls.
Right of refusal at providing registry	Providing registry	A providing registry receiving a forwarded query may inspect the

		originating entity’s forwarded manifest scope and refuse the request if incompatible with the providing registry’s access policy.

4. Service Invocation Authentication

Service invocation authentication governs how a requesting agent establishes trust with a selected service agent at runtime. This section is distinct from Sections 1–3, which govern contexts where the registry is an active participant. At invocation time, the registry is not in the path.

The registry's role ends at discovery: it surfaces the service's invokable endpoint and OIDC endpoint as part of the manifest returned by searchAndInvoke. All subsequent authentication and invocation occurs directly between the requesting agent and the service agent. The registry is neither contacted nor involved for the duration of the invocation session.

> *The authentication lifecycle described in this section is encapsulated by the searchAndInvoke helper. Calling agents interact with a single invocation interface and do not manage the mechanics below directly.*

4.1 The Server as Sovereign Token Issuer

SADAR does not designate a central token authority for service invocation. Each service agent is the sovereign issuer of its own usage tokens. This design:

- Keeps the registry out of the runtime invocation path
- Ensures each service maintains sovereign control over its own access decisions and policy
- Eliminates single points of failure in the authentication chain
- Allows each service to apply its own governance policy at token issuance, independent of any central authority

This stands in contrast to registry authentication (Section 3), where the registry is the token issuer for query session tokens. At invocation time, the service issues the token; the registry's prior contribution was surfacing the service's OIDC endpoint at discovery.

4.2 Manifest Fields Supporting Service Invocation Authentication

Every SADAR-compliant service manifest carries the following fields that are directly relevant to invocation authentication. All required fields are covered by the manifest's JWS signature; any post-upload alteration is detectable by any validating party.

Field	Required	Purpose at Invocation
invokable_endpoint	Required	The direct URI at which the service agent accepts invocation requests. Used by searchAndInvoke for all runtime calls. Surfaced by the registry at discovery time; the registry is not involved at invocation time.
oidc_issuer	Required	URI identifying the service agent's OIDC authority. Serves as the root from which the requester derives the service's token endpoint. Also serves as the stable identifier for the

		service's identity domain. Present in the requester's manifest to identify the requester's own OIDC authority to the server.
jwks_uri or jwks	Required	The entity's public key set. Serves three functions: (1) verifying the manifest's JWS signature; (2) validating the JWT presented at first-use provisioning; (3) verifying DPoP proof signatures on every subsequent request.
a2a_card_uri	Optional	URI pointing to the service agent's A2A Agent Card at its well-known endpoint. Navigational only — for A2A-native clients who want the full A2A interaction contract. SADAR-aware clients use invokable_endpoint directly.

4.3 The Signed Manifest as Identity Anchor

A SADAR manifest is publicly accessible with its hash. Possession of the manifest alone proves nothing — any party can retrieve and present a public document. The manifest's function as an identity artifact depends on combining it with runtime proof-of-key-possession.

The signing key creates the binding: the manifest is signed with the entity's private key (JWS/ES256); the manifest's jwks field contains the corresponding public key; at runtime, the entity presents a JWT also signed with its private key; the relying party verifies the JWT against the JWKS derivable from the manifest. Successful verification proves the presenter controls the key that signed the manifest — i.e., is the entity the manifest declares.

This means the JWKS in the manifest is load-bearing for three distinct verification operations: manifest signature verification, JWT validation at first-use provisioning, and DPoP proof verification on every subsequent request. One key set serves all three functions without additional key material.

4.4 Identity Granularity and Credential Mapping

The direct model for service invocation authentication — in which every requesting agent registers individual credentials at every service it calls — creates a credential management problem that scales as the product of the number of requesters and the number of services. SADAR resolves this through a registry-mediated indirection: the credential used to authenticate a request is not bound directly to the individual agent, but to a configured identity scope that may represent an entity, a project, a division, or the individual agent. The registry's credential_scope field on each agent registry entry defines this mapping.

When searchAndInvoke makes the OIDC authentication call on behalf of an agent, it presents the agent's registry identity. The OIDC server resolves the credential_scope mapping and issues a token reflecting the configured scope — not necessarily the individual agent. The service being called therefore registers a trust relationship with the issuer endpoint of the mapped identity (an entity, project, or other scope) rather than with each individual agent. Multiple agents within the same entity or project share a single registered trust relationship at each service, without sharing credentials.

The credential_scope field accepts the following configured values:

- agent — the authentication credential is bound to the individual agent registry entry. Services register a trust relationship with this agent's identity specifically. This scope is

appropriate for agents with highly differentiated access requirements or regulatory obligations that mandate agent-level accountability.

- entity — all agents within the entity share a single service-facing identity. Services register a trust relationship with the entity's OIDC issuer once; all agents within that entity are authenticated under the entity's credential.
- project — agents grouped under a common project or business process share a single service-facing identity. Useful for organizing trust relationships by functional boundary rather than organizational boundary.
- division — agents within an organizational division or business unit share a service-facing identity scoped to that division.

The credential_scope mapping is an administrative property of the agent's registry entry, established at provisioning time and not modifiable at runtime by the calling application. The scope of the issued authentication token is determined by the registry; the calling application cannot assert a different scope. The individual agent's identity within that scope — and all other authorization context — is carried separately in the SADAR Context Token (see Section 4.6).

4.5 Agent Credential Isolation

SADAR is designed to overlay any agentic framework without requiring framework-specific credential management code in the agent itself. Agents are not expected to know how to authenticate; authentication is an infrastructure concern managed by searchAndInvoke on the agent's behalf. The governing security property that follows from this design principle is agent credential isolation: the private credential used to authenticate on behalf of an agent must not be directly accessible to the agent's application logic.

This requirement does not prescribe how isolation is achieved. It constrains only the outcome: the credential must not be retrievable by code executing within the agent's own logic boundary. searchAndInvoke, operating in the agent's infrastructure layer rather than its application layer, accesses the credential via the deployment's provisioned trust boundary and uses it to produce a signed assertion. The agent's application code never touches the credential material.

The normative requirements for credential binding and isolation are stated in Appendix A. Non-normative guidance on conformant implementation patterns (platform workload identity, credential broker sidecar, and framework-level injection) is provided in Appendix A, Section A.4.

Each conformant agent deployment is provisioned with a private credential at registration time, prior to any invocation. The credential is scoped to the individual agent registry entry regardless of the credential_scope mapping — the scope mapping governs how the agent's identity is presented to services, not how many credentials are provisioned. An agent whose credential_scope is entity still has its own provisioned credential; the entity mapping is resolved by the OIDC server at token issuance, not by sharing credentials across agents.

4.6 SADAR Context Token (SCT)

The OIDC JWT obtained during first-use provisioning establishes authentication at the configured credential_scope granularity. It answers the question of who is permitted to call the service at the infrastructure level. It does not carry the full context of what is being requested, why, on whose authority, and as part of what business process. That context is carried in the SADAR Context Token (SCT).

The SCT is a signed authorization context token constructed by searchAndInvoke and transmitted alongside the authentication token on every invocation. It carries claims that the

authentication layer deliberately does not carry: the specific agent identity within the mapped scope, the originating user's identity (whether directly authenticated or asserted by the calling application), the business process identifier, the current step within that process, the authorized operation types, the per-step risk adjustment and step status, the process-instance identifier, the scope of authority, and the chain-of-custody linkage to any parent invocation. The SCT is what enables a service to enforce fine-grained policy beyond "this entity is permitted to call me."

The separation between the authentication layer (OIDC JWT) and the authorization context layer (SCT) is intentional. The authentication layer is stable across an invocation session and represents a registered trust relationship. The authorization context layer is specific to each invocation and represents the runtime assertion of what is being done. Keeping them separate prevents the authentication credential from becoming overloaded with policy context that varies per call, and allows the credential_scope mapping to operate independently of the authorization policy enforced at each service.

The SCT is constructed and signed by searchAndInvoke. Its claims include:

- agent_id — the specific agent registry URN, regardless of the credential_scope mapping. The service always knows which agent is the actual caller, even when the authentication token presents a broader scope.
- originating_user — the identity of the human user whose session initiated the invocation chain. May be a directly authenticated user JWT (if the calling application passed user credentials) or an asserted user claim (if the application launched the agent under its own service credentials and vouches for the user identity). The SCT carries a flag indicating which trust model applies.
- business_process_id — the SADAR-registered identifier of the business process under which this invocation occurs. Services may enforce policy based on whether the invocation is part of a recognized, authorized business process.
- agent_step — the step identifier within the business process at which this invocation occurs. Used by services enforcing step-level operation constraints.
- authorized_operations — the set of operation types this invocation is authorized to request, as declared in the agent's manifest for this step. Services enforce that requested operations fall within this set.
- parent_sct_jti — the jti of the SCT from the parent invocation in the call chain, if any. Enables full chain-of-custody reconstruction in audit logs.
- risk_score_adjustment — the step's net contribution to the risk score, carried in the clear (signed, not encrypted) as { delta, reason }. delta is a decimal in [-1.0, 1.0]; reason is an IRI, normatively from the SADAR-defined urn:sadar:risk_reason:v1:* enumeration (custom reasons in an organization-controlled namespace). Free-form rationale is recorded in OTel telemetry, where it may be encrypted, and never in this clear claim. The accumulated scalar is carried in OTel baggage as a convenience and is recomputable from the signed chain; see 13. Risk Score Specification.
- step_status — the terminal outcome the step reports for this invocation, carried in the clear and mandatory on every segment. A structure of { status, reason, description }. status is an IRI from urn:sadar:step_status:v1:* (success, success_with_information, success_with_warnings, partial_success, failure, failure_compensated, failure_uncompensated, cancelled; default success). reason is an IRI from urn:sadar:status_reason:v1:* and is required for any non-clean outcome. description is sanitized free text (≤ 256 characters; the set [A-Za-z0-9], space, and . , - _ : only; markup, quotes, brackets, URLs, and control characters are stripped on ingestion); it is

display-only and SHALL NOT be parsed or used in any control decision. The step reports; the enforcement layer decides. Longer or sensitive narrative is recorded in OTel telemetry, where it may be encrypted, and never in this clear claim.

- intent_instance_id — the stable identifier of this process instance. Seeded from the root TraceID at startFlow and propagated by value, it is immutable end-to-end and identical across trust boundaries. It is the join key that correlates repatriated telemetry to the originating flow. It is distinct from the local TraceID (which may legitimately differ in a remote segment) and SHALL NOT be re-read from the ambient trace downstream.
- authority — the scope of authority for this invocation, structured as RFC 9396 (OAuth 2.0 Rich Authorization Requests) authorization_details with the SADAR type urn:sadar:authority:v1: actions (operations), datatypes (data scope), locations (targets and resources), and privileges (level). The authorized_operations above are the actions component of this structure. SADAR does not set, compute, interpret, or attenuate this authority; an IAM or policy or authorization server establishes it and the enforcement layer checks it, while the SCT propagates it, signed. Authority always travels in the SCT regardless of how the caller authenticated (OAuth/OIDC, SPIFFE SVID, or mTLS).
- cnf — a confirmation (proof-of-possession) binding to the authentication credential presented on this invocation (for example a DPoP key thumbprint, a SPIFFE SVID, or an mTLS client certificate), so that a captured SCT cannot be replayed under a different caller. It cryptographically ties authentication (the auth token) to authority (this SCT).
- segment_action — the action this segment records, carried in the clear (signed, not encrypted) as an IRI from the SADAR-defined urn:sadar:segment_action:v1:* enumeration. The reference values are urn:sadar:segment_action:v1:executed (a component performed its operation; the default for an ordinary execution segment), urn:sadar:segment_action:v1:routed (a registry gateway received an inbound cross-boundary call and routed it internally; signed with the home registry's key), and urn:sadar:segment_action:v1:executed_nonconformant (searchAndInvoke executed a registry-defined operation against a non-conformant agent, tool, or resource that cannot itself sign; signed with the home registry's key on the registry's behalf). The enumeration is extensible; additional actions may be added in subsequent spec versions, and implementations ignore action IRIs they do not recognize for control purposes while preserving them for audit. The action makes the role of a segment explicit rather than inferred, and disambiguates the two registry-signed segment kinds (routed versus executed_nonconformant). See the SCT Propagation companion for the boundary-crossing and conformance semantics.

The SCT is transmitted as a signed JWS-inside-JWE in a dedicated SADAR-SCT HTTP header on every invocation, alongside the OIDC usage token in the Authorization header rather than embedded within it. Services validate the SCT signature using the signing key declared for the searchAndInvoke implementation in use. The SCT is not a replacement for the DPoP-protected usage token; both are required on every request. The usage token provides authentication and replay protection; the SCT provides authorization context and chain-of-custody.

4.7 First-Use Provisioning

When a requesting entity invokes a service agent for the first time, a provisioning exchange establishes the bilateral trust relationship. This exchange is orchestrated transparently by searchAndInvoke:

- searchAndInvoke authenticates to the requester's OIDC token endpoint using a signed JWT assertion (RFC 7523 — JSON Web Token Profile for OAuth 2.0 Client Authentication). The assertion is signed with the agent's provisioned private credential, accessed via the agent's trust boundary as described in Section 4.5. The credential is never passed through agent application logic. The OIDC server validates the assertion against the agent's registered public key and issues a JWT whose identity scope reflects the agent's credential_scope registry mapping (see Section 4.4).
- searchAndInvoke presents the JWT plus a manifest reference (Entity URN + oidc_issuer URI) to the service agent's invokable endpoint.
- The service agent resolves the requester's oidc_issuer URI to its discovery document ({issuer}/.well-known/openid-configuration) and retrieves the jwks_uri.
- The service fetches the requester's JWKS and validates the JWT signature. Proof-of-key-possession confirmed: the presenter controls the private key corresponding to the JWKS public key, which is the same key that signed the manifest.
- The service validates JWT claims: iss must match the oidc_issuer declared for the mapped identity; sub must match the identity established by the agent's credential_scope mapping — which may be an entity URN, project URN, or other configured granularity rather than the individual agent URN. The service does not require the individual agent URN in the authentication credential.
- The service provisions access for the Entity URN and issues a usage token bound to that URN.
- searchAndInvoke caches the usage token for subsequent calls to this service.

After successful first-use provisioning, the service may cache the requester's JWKS for the duration of the manifest's declared TTL, refreshing on manifest version change or TTL expiry. The service does not call the requester's OIDC endpoint again unless the requester's manifest changes — subsequent authentication uses the cached usage token.

4.8 Runtime Authentication Flow

After first-use provisioning, subsequent invocations from the same requesting entity are authenticated using the cached usage token:

- searchAndInvoke presents the cached usage token as a Bearer token in the Authorization HTTP header.
- A DPoP proof accompanies each request (see Section 4.9).
- The service validates the usage token and DPoP proof before processing the request.
- Token refresh: proactive refresh when exp is within a configurable threshold (default 30 seconds); reactive refresh on 401 response. The service's OIDC endpoint is called only at initial token acquisition and token refresh — not on every request.

4.9 DPoP Replay Protection

Standard JWT Bearer tokens are not inherently replay-proof: a captured token remains valid until its expiry timestamp. SADAR mandates DPoP — Demonstration of Proof of Possession (RFC 9449) — to eliminate this window.

For each request, the requester generates a DPoP proof: a short-lived JWT containing the HTTP method and URI of the specific request (htm and htu claims), a unique jti not previously seen by this service, an iat timestamp with a short validity window, and signed with the requester's private key.

The service validates all of the following before processing:

- The access token (standard Bearer token validation)
- The DPoP proof signature, using the same JWKS used at first-use provisioning
- That the jti has not been seen before within the expiry window (the service maintains a jti seen-set)
- That the htm and htu claims match the actual request method and URI

A captured usage token is useless without a valid DPoP proof. A captured DPoP proof is useless because its jti is already in the seen-set and its htm/htu are bound to a specific request. The combination provides per-request, method-and-URI-bound replay protection without a challenge-response round trip. The requester's JWKS — derivable from the manifest — serves all three verification functions: manifest signature, first-use JWT validation, and per-request DPoP proof. No additional key material is required.

4.10 A2A Protocol Interoperability

The SADAR authentication model is fully compatible with the Agent2Agent (A2A) protocol's authentication declaration model. A2A Agent Cards declare authentication requirements using OpenAPI 3.2 Security Scheme Objects, which support the same schemes SADAR uses: openIdConnect, oauth2 (Client Credentials flow), and mtls. The service's OIDC endpoint declared in the SADAR manifest corresponds directly to the openIdConnectUrl in an A2A Agent Card's openIdConnectSecurityScheme — they reference the same identity authority.

A2A explicitly requires that credential acquisition be out-of-band — the A2A protocol does not define how clients obtain tokens. searchAndInvoke is that out-of-band process, standardized by SADAR. There is no conflict between the two models; they operate at complementary layers.

4.10.1 The a2a_card_uri Field

A SADAR manifest may optionally carry an a2a_card_uri field pointing to the service agent's A2A Agent Card at its well-known endpoint (e.g. https://agent.example.com/.well-known/agent-card.json). The scope of this field is strictly navigational:

- SADAR-aware clients use invokable_endpoint in the manifest directly. No additional hop to the A2A card is required.
- A2A-native clients who encounter the manifest via a registry query may follow the a2a_card_uri to retrieve the full A2A interaction contract — skills, streaming capabilities, push notification support, and other A2A-specific interaction details not carried in the SADAR manifest.
- The a2a_card_uri also serves as a verifiable cross-reference: a consumer can confirm that the SADAR manifest and the A2A Agent Card describe the same service agent.

> *There is no change to the A2A Agent Card itself. SADAR does not modify, extend, or replace the A2A card. The a2a_card_uri in the SADAR manifest is a one-way pointer from the SADAR discovery artifact to the A2A invocation artifact. The SADAR registry stores manifests. It does not store A2A Agent Cards.*

4.10.2 Relationship to A2A Discovery

A2A's own discovery model requires the client to already know the agent's domain before fetching its card from /.well-known/agent-card.json. SADAR provides the prior step: finding which agent to contact in the first place, based on semantic capability matching via searchAndInvoke. SADAR is the semantic discovery layer that A2A explicitly leaves undefined. Once discovery resolves a service, the invocation proceeds — via SADAR's invokable_endpoint or via A2A's interaction contract, at the client's discretion.

4.11 searchAndInvoke Token Management Summary

The searchAndInvoke helper encapsulates the full invocation authentication lifecycle:

- Retrieves the service's OIDC endpoint and invokable endpoint from the manifest returned by discovery
- Calls the requester's own OIDC endpoint to obtain the initial JWT for first-use provisioning
- Manages the first-use provisioning exchange with the service agent
- Caches the issued usage token per service endpoint
- Generates and attaches a DPoP proof to each request (RFC 9449)
- Proactively refreshes the usage token when exp is within a configurable threshold (default: 30 seconds)
- Handles reactive token refresh on 401 responses
- Optionally follows a2a_card_uri for A2A-native invocation when requested

The calling agent interacts with a single invocation interface. It does not manage authentication tokens, DPoP proofs, OIDC endpoints, or JWKS resolution directly.

4.12 Accountability Table

Control	Owner	Mechanism / Notes
Requester private key custody	Requesting entity	Never leaves the requesting entity's environment. Used to sign the JWT presented at first-use provisioning and to generate DPoP proofs on every request.
Requester manifest integrity	Requesting entity + local registry	JWS signature over manifest content, including oidc_issuer and jwks fields. The signed manifest binds the Entity URN to the OIDC issuer URI and public key. Post-upload tampering is detectable by any validating party.

First-use identity verification	Service agent	Service fetches requester's JWKS via oidc_issuer discovery document and validates the presented JWT signature. Proof-of-key-possession confirms the presenter controls the key that signed the manifest.
Usage token issuance	Service agent	The service is the sovereign issuer of its own usage tokens. No central token authority. Each service applies its own access policy at token issuance.
Usage token caching	searchAndInvoke helper	Caches issued usage token per service endpoint. Proactively refreshes when exp is within a configurable threshold (default 30 seconds).
Replay protection	searchAndInvoke + service agent	DPoP (RFC 9449): per-request proof JWT bound to HTTP method, URI, jti, and iat. Service maintains jti seen-set within expiry window. Captured tokens and proofs cannot be replayed.
Token expiry and refresh	searchAndInvoke helper	Proactive refresh via silent re-authentication to the service's OIDC endpoint. Reactive refresh on 401 response.
A2A interoperability	searchAndInvoke helper	searchAndInvoke is the out-of-band credential acquisition process that A2A explicitly leaves undefined. The service's oidc_issuer in the SADAR manifest corresponds to the openIdConnectUrl in an A2A Agent Card security scheme.

Appendix A: Normative Requirements — Agent Credential Binding and Invocation Authentication

This appendix states the normative requirements that flow from the architectural design described in Sections 4.4, 4.5, and 4.6. Requirements are expressed as SHALL (mandatory), SHOULD (strongly recommended), and SHALL NOT (prohibited). Conformance is assessed against these statements, not against specific implementation mechanisms. Non-normative implementation guidance is provided in Section A.4.

A.1 Agent Credential Binding

- A.1.1 — Each registered agent entity SHALL have an associated authentication credential bound to its SADAR registry identity at provisioning time. This credential SHALL be the basis for all authentication assertions made on behalf of that agent.
- A.1.2 — The credential binding SHALL be established prior to any agent invocation. Credential binding during an active invocation transaction is not permitted.

- A.1.3 — The credential SHALL NOT be directly accessible to agent application logic. Conformant implementations MUST ensure that the credential is not retrievable by code executing within the agent's own logic boundary.
- Note: The mechanism by which credential isolation is achieved is an implementation concern. Conformant approaches include but are not limited to: platform-attested workload identity (e.g. cloud provider managed identity services), credential broker sidecar services exposing a local loopback signing endpoint, and framework-level injection prior to agent execution. See Section A.4 for non-normative guidance on each pattern.
- A.1.4 — The credential SHALL be scoped to a single registered agent identity. Credentials shared across multiple distinct agent registry entries do not satisfy this requirement.
- Note: Where the registry credential_scope field resolves an agent identity to a broader scope (entity, project, division), the credential itself remains agent-scoped. The broader scope is a property of the identity mapping at the OIDC server, not of credential sharing between agents. Multiple agents may present the same service-facing identity through the mapping without sharing private credentials.

A.2 Invocation Service Authentication

- A.2.1 — The SADAR Invocation Service acting on behalf of a registered agent SHALL authenticate to the target service's authorization server using a signed assertion that demonstrates binding to that agent's registered credential.
- A.2.2 — The signed assertion SHALL conform to RFC 7523 (JSON Web Token Profile for OAuth 2.0 Client Authentication and Authorization Grants) or an equivalent mechanism that provides equivalent non-repudiation and replay protection properties.
- A.2.3 — The Invocation Service SHALL NOT transmit the agent's private credential material over any network boundary to obtain an authentication token. The credential SHALL be used only to produce a signed assertion within the agent's trust boundary.
- A.2.4 — The authentication token returned by the authorization server SHALL reflect the identity granularity defined by the agent's registry credential_scope mapping. The mapping SHALL resolve to one of the following configured scopes: agent, entity, project, or division. The resolved scope SHALL be established in the registry and SHALL NOT be determined at invocation time by the calling application.
- A.2.5 — The Invocation Service SHALL NOT accept caller-supplied credential material as a substitute for the registered agent credential. The credential used for authentication SHALL be obtained exclusively from the agent's provisioned trust boundary.

A.3 Credential Lifecycle

- A.3.1 — Conformant implementations SHALL support credential rotation without requiring re-registration of the agent's SADAR registry entry.
- A.3.2 — Revocation of an agent credential SHALL be propagatable to all authorization servers with which that credential's mapped identity is registered within a time window defined by the deployment's security policy.
- A.3.3 — Credential expiry and rotation SHALL be managed outside the agent invocation path. An agent invocation SHALL NOT trigger credential provisioning.

A.4 Non-Normative Implementation Guidance

The following patterns represent conformant approaches to satisfying the requirements in Sections A.1 through A.3. Implementors are not required to use any specific pattern; any approach that satisfies the normative requirements is conformant. Patterns are listed in order of preferred security posture.

A.4.1 Platform Workload Identity (Preferred)

The deployment platform attests the workload's identity at launch and makes a signed token available via a local metadata endpoint. searchAndInvoke calls the local endpoint to obtain a token representing the agent's identity; no private key is ever retrieved by application or infrastructure code. Examples include cloud provider managed identity services and SPIFFE/SPIRE workload identity frameworks. This pattern satisfies A.1.3 architecturally: the private key does not exist as a retrievable artifact in the application environment.

A.4.2 Credential Broker Sidecar

A minimal sidecar process co-deployed with the agent holds the agent's private credential and exposes only a local loopback signing endpoint. searchAndInvoke submits a claims payload to the sidecar and receives a signed assertion in return; it never receives or holds the private key. The sidecar's secrets manager access is scoped to the single agent credential by deployment configuration. This pattern is appropriate for deployment targets that do not support platform workload identity and provides a comparable blast-radius boundary: a compromise of the sidecar exposes one agent's credential, not all agents' credentials.

A.4.3 Framework-Level Injection

In framework environments that support pre-invocation hooks (decorators, middleware, or explicit setup calls prior to agent execution), the deployment developer retrieves the agent's private credential from a secrets manager and injects it into a session-scoped context accessible to searchAndInvoke. The credential is held in a controlled in-memory area for the duration of the session. searchAndInvoke reads from this context rather than from the secrets manager directly. The developer is accountable for secrets manager access; searchAndInvoke receives the credential via the injected context rather than fetching it independently. This pattern satisfies A.1.3 when the injection occurs prior to agent logic execution and the context is not accessible to agent application code. It is appropriate only where the framework provides a reliable pre-execution injection point that the agent cannot subvert.

A.4.4 Conformance Priority

Implementations SHOULD prefer platform workload identity where the deployment target supports it. Where platform workload identity is unavailable, a credential broker sidecar is the recommended alternative. Framework-level injection is permitted where neither of the preceding patterns is applicable. Implementations that give searchAndInvoke direct, unrestricted access to a secrets manager containing credentials for multiple agents are permitted but not recommended; such implementations SHOULD document the scope restriction applied to limit searchAndInvoke's secrets manager access to the agents it is authorized to represent.