AGEX FOUNDATION | PREPRINT | arXiv:cs.AI | 19 February 2026

AGEX

Agent Gateway Exchange, eXtensible

The Autonomous Credential Protocol for the Agentic Web

Hayley Brown **Mike Ensor**

AGEX Foundation

agex.api

19 February 2026

Abstract

The proliferation of autonomous AI agents has created a fundamental and growing crisis in digital trust infrastructure. Existing authentication protocols, designed for human-delegated application access, are structurally incapable of supporting the speed, autonomy, and credential lifecycle requirements of AI agent systems operating at machine scale. Current workarounds including hardcoded credentials, shared service accounts, and manually rotated API keys introduce systemic security vulnerabilities that scale linearly with agent deployment size and have no principled resolution within existing protocol frameworks. This paper introduces AGEX (Agent Gateway Exchange, eXtensible), a new open protocol specification for autonomous credential management in AI agent systems. AGEX defines a standardised framework through which AI agents can discover, request, receive, rotate, escalate, delegate, and revoke API credentials autonomously, without human intervention within pre-authorised parameters. The protocol introduces five new cryptographic primitives: Agent Identity Documents (AIDs), Intent Manifests, Credential Lifecycle Contracts (CLCs), the Autonomous Rotation Protocol (ARP), and Delegation Chains. Together these primitives form a complete, auditable, and revocable trust infrastructure for the agentic web. We present the full protocol specification including complete HTTP API definitions, message schemas with field-level normative descriptions, wire format examples, state machine specifications, cryptographic constructions, formal security analysis with proofs, pseudocode for all protocol flows, a comparative evaluation against OAuth 2.0, OpenID Connect, and SPIFFE/SPIRE, the AGEX Policy Language specification, the Hub Federation Protocol, six worked deployment scenarios, conformance testing requirements, and a comprehensive privacy and compliance analysis. We demonstrate that AGEX satisfies the security properties of credential confidentiality, forward secrecy under rotation, delegation scope monotonicity, and cascade revocation completeness under standard cryptographic assumptions. We propose the AGEX Foundation as the open governance body for the specification and describe an implementation roadmap targeting IETF standardisation as a Proposed Standard RFC.

Keywords: *AI agents, autonomous authentication, credential lifecycle management, protocol design, OAuth, API security, zero-trust, key rotation, delegation, agentic systems, IETF, cryptographic protocols, identity management, authorisation*

Table of Contents

1. Introduction

1.1 The Agentic Transition

Software is undergoing a structural transition that has no precise historical precedent. For four decades, computer programs executed deterministic instructions authored entirely by human developers. Every action a program could take was explicitly specified at development time; the set of possible program behaviours was bounded by human foresight. The emergence of large language model-based AI agents marks a fundamental departure from this model.

AI agents are software systems that plan, reason, and act dynamically in pursuit of goals. They decompose objectives into sub-tasks at runtime, select appropriate tools, call external APIs, read and write data, spawn sub-processes, and adapt their behaviour based on intermediate results, all without deterministic pre-specification of every action. The set of possible behaviours of a sufficiently capable agent is not bounded by developer foresight; it is bounded only by the agent's access to external systems and the scope of its authorised permissions.

This transition is not theoretical. Production deployments of AI agent systems are operating across industries including finance, logistics, healthcare, legal services, and digital marketing. Frameworks including LangChain, AutoGen, CrewAI, and n8n have lowered the barrier to building agents that interact with dozens of external services simultaneously. Enterprise adoption is accelerating. Analyst estimates place the number of production AI agent deployments in the tens of thousands as of early 2026, with projections reaching millions within three years.

1.2 The Credential Management Crisis

Every agent action that touches an external service requires a credential: an API key, a bearer token, a session secret, an OAuth grant, a database password, a webhook signing key. The management of those credentials, specifically how they are issued, stored, transmitted, rotated, delegated, audited, and revoked, is the foundational security challenge of the agentic web. It is currently unsolved at the protocol level.

In the absence of a purpose-built protocol, development teams have adopted credential management practices that introduce serious and compounding security risks. The most common patterns observed in production deployments include the following.

Static API keys embedded in agent code or environment configuration files represent the dominant pattern. These keys have no rotation schedule, no usage auditing at the protocol level, and no revocation mechanism tied to agent lifecycle events. When an agent is decommissioned, its credentials frequently persist in configuration management systems, version control history, and deployment artefacts indefinitely. Security audits of large-scale agent deployments routinely discover credentials that were issued for agents decommissioned 12 to 24 months previously but remain valid and unrevoked.

Shared service accounts represent the second most common pattern. Multiple agents, sometimes hundreds operating in parallel, share a single credential identity. This pattern makes attribution of individual actions to specific agents impossible, makes targeted revocation impossible without disrupting all agents sharing the identity, and violates the principle of least privilege at scale. A breach of one agent in a shared-credential deployment compromises all agents sharing that identity simultaneously.

Over-permissioned credentials arise because the operational overhead of managing fine-grained, task-specific permissions is prohibitive with current tooling. Agents are therefore routinely granted maximum available scope. The result is that a compromised agent in a read-only data analysis role may hold write credentials to production databases, payment APIs, and customer communication systems.

Manual rotation cycles managed by security teams on fixed schedules, typically quarterly or annually, create predictable vulnerability windows. Credentials compromised between rotation cycles may be exploited for months before revocation. Coordinating rotation across hundreds of agents and dozens of services simultaneously is an operational challenge that frequently results in service disruption, causing teams to extend rotation intervals to reduce operational risk, which further widens vulnerability windows.

No cross-service revocation infrastructure means that when an agent is compromised, there is no standard mechanism to revoke its access across all connected services simultaneously. Security teams must manually identify and revoke credentials service by service, a process that may take hours or days for large deployments, during which time the compromised agent retains full access to all services it was authorised to reach.

These patterns are not implementation failures by negligent engineering teams. They are the rational response of competent engineers to the complete absence of a protocol that addresses their actual operational requirements.

1.3 Contributions

This paper makes the following technical contributions.

First, we formally characterise the structural incompatibilities between existing authentication protocols (OAuth 2.0, OpenID Connect, SPIFFE/SPIRE, and API key schemes) and the operational requirements of autonomous AI agents. We demonstrate that these incompatibilities cannot be resolved through incremental extension of existing standards without fundamental redesign of their core abstractions, and we quantify this claim through a twelve-criterion comparative evaluation.

Second, we introduce AGEX (Agent Gateway Exchange, eXtensible), a complete protocol specification for autonomous agent credential lifecycle management. AGEX is specified at the level of HTTP API endpoints with complete request and response schemas, state machine definitions for all protocol participants, wire format examples for every message type, and normative language (MUST, SHOULD, MAY per RFC 2119) throughout.

Third, we define five new protocol primitives: Agent Identity Documents (AIDs) establishing cryptographically verifiable agent identity; Intent Manifests providing structured semantic declarations of agent purpose; Credential Lifecycle Contracts (CLCs) governing the complete credential relationship; the Autonomous Rotation Protocol (ARP) enabling agent-initiated key rotation without human intervention; and Delegation Chains providing cryptographically enforced scope-reducing credential inheritance for multi-agent systems.

Fourth, we present a formal security analysis demonstrating that AGEX satisfies eight defined security requirements under standard cryptographic assumptions. For each requirement, we provide a theorem statement, proof sketch, and analysis of the conditions under which the requirement may be violated.

Fifth, we specify the AGEX Policy Language (APL), a JSON-based declarative language for expressing credential approval policies, enabling service providers to define automated approval rules that correctly handle the full range of agent trust tiers and intent types without requiring human review for routine requests.

Sixth, we specify the AGEX Hub Federation Protocol (AHFP), defining how multiple AGEX Hub instances interoperate to support cross-organisational agent credential requests.

Seventh, we present six worked deployment scenarios demonstrating AGEX applied to representative real-world agent architectures, including single-agent API access, multi-agent orchestration, cross-organisational agent federation, emergency incident response, regulatory compliance workflows, and consumer-facing agent services.

Eighth, we define a conformance testing framework specifying the minimum requirements for an implementation to be considered AGEX-conformant, enabling interoperability verification across independent implementations.

Ninth, we present a comprehensive privacy and compliance analysis addressing GDPR, CCPA, data residency requirements, and the intersection of agent credential management with emerging AI governance frameworks.

1.4 Paper Organisation

Section 2 surveys related work and establishes the limitations of existing protocols. Section 3 presents the threat model and security requirements. Section 4 defines the five AGEX protocol primitives with full schemas. Section 5 specifies the HTTP API for all Hub endpoints. Section 6 defines the AGEX Policy Language. Section 7 describes the Hub architecture and federation protocol. Section 8 provides pseudocode for all protocol flows. Section 9 presents the formal security analysis. Section 10 presents the comparative evaluation. Section 11 presents six worked deployment scenarios. Section 12 defines the conformance testing requirements. Section 13 provides a privacy and compliance analysis. Section 14 describes governance and the implementation roadmap. Section 15 concludes.

2. Background and Related Work

2.1 OAuth 2.0 and OpenID Connect

OAuth 2.0, standardised as IETF RFC 6749 in October 2012 [RFC6749], is the dominant protocol for delegated authorisation on the web. The framework defines four grant types for different interaction patterns. The authorisation code grant targets web applications where a human user authorises a client via a browser redirect. The implicit grant was designed for browser-based applications (now deprecated in favour of authorisation code with PKCE per RFC 7636). The resource owner password credentials grant allows a client to use a user's credentials directly (deprecated due to security concerns). The client credentials grant allows a client to authenticate using its own credentials, representing the OAuth 2.0 flow most applicable to machine-to-machine scenarios.

The client credentials grant is the OAuth 2.0 flow most commonly applied to agent authentication in production deployments today. It allows an application to authenticate directly without a human user in the authorisation flow. However, it suffers from five structural limitations in agent contexts that cannot be resolved without fundamental changes to the grant type.

The client secret is static. There is no standardised mechanism within the client credentials grant for rotating the client secret without manual intervention and a service disruption window. RFC 7591 (Dynamic Client Registration) partially addresses this by allowing programmatic client registration, but does not define automated rotation of registered client credentials.

Scope is fixed at registration time. The scopes available to a client credentials grant are determined when the client is registered with the authorisation server. There is no standardised mechanism for an agent to request a different scope set for a specific task without re-registering as a new client or using a separate client identity for each scope combination.

There is no delegation primitive in the client credentials grant. A client cannot create sub-credentials for downstream processes with constrained scope. In multi-agent systems, each agent must independently hold its own client credentials, and the relationship between an orchestrator agent and its worker agents has no representation in the protocol.

There is no lifecycle management beyond token expiry. The client credentials grant produces access tokens with a defined expiry, renewed via a new client credentials request. There is no concept of a credential relationship with defined rotation obligations, audit requirements, or conditional termination.

There is no semantic intent declaration. The authorisation request carries a scope parameter (a space-delimited list of strings) but no mechanism for declaring why access is being requested, what the agent intends to do with the access, or what the acceptable use constraints are. This makes automated policy evaluation difficult and makes post-hoc audit interpretation reliant on log correlation rather than structured intent records.

Subsequent OAuth 2.0 extensions address some of these limitations but do not resolve them comprehensively. Pushed Authorisation Requests (PAR, RFC 9126 [RFC9126]) improve security for authorisation code flows but do not apply to client credentials. Rich Authorisation Requests (RAR, RFC 9396 [RFC9396]) allow more structured authorisation data in the request, providing a partial basis for intent declaration, but define no standard schema for agent intent and provide no lifecycle management. Token Binding (RFC 8471 [RFC8471]) binds tokens to TLS connections but does not address rotation, delegation, or lifecycle management. JWT-based client authentication (RFC 7523 [RFC7523]) allows a private key JWT assertion instead of a client secret, improving the security of the client credential but not addressing any of the five structural limitations listed above.

OpenID Connect (OIDC), published as a specification by the OpenID Foundation in 2014 and widely adopted as the de facto standard for web authentication, defines an identity layer on top of OAuth 2.0. OIDC adds the ID token, a JSON Web Token (JWT) containing claims about the authenticated entity, and the UserInfo endpoint, which returns additional claims.

OIDC's primary contribution to the identity landscape is the standardisation of identity assertion format and the federation of identity providers. These contributions are significant for human identity but map poorly to agent contexts for the following reasons.

The ID token is designed to assert the identity of a human user who has actively authenticated. Its standard claims (sub, name, email, phone_number, birthdate, and so forth) have no meaningful analogue for an autonomous agent. Attempts to repurpose OIDC for agent identity require wholesale replacement of the standard claim set, at which point the resulting structure is no longer OIDC in any meaningful sense.

The OIDC discovery mechanism (RFC 8414 [RFC8414]) provides a standardised way for clients to discover authorisation server metadata. This pattern is valuable and AGEX adopts an analogous service provider discovery mechanism. However, the OIDC discovery document is designed for human-facing authorisation flows and does not carry the agent capability metadata, trust tier information, or policy endpoint references that AGEX requires.

OIDC does not address credential lifecycle management, rotation, delegation, or cascade revocation. It inherits all five structural limitations of the OAuth 2.0 client credentials grant and adds no agent-relevant capabilities beyond the JWT-formatted identity assertion.

2.2 SPIFFE and SPIRE

SPIFFE (Secure Production Identity Framework for Everyone) and its reference implementation SPIRE (SPIFFE Runtime Environment) represent the most significant prior work relevant to AGEX. Developed within the Cloud Native Computing Foundation (CNCF) and published as a specification in 2021 [SPIFFE], SPIFFE addresses the problem of workload identity in distributed systems.

SPIFFE defines two key constructs. The SPIFFE ID is a URI identifying a workload within a trust domain, taking the form spiffe://trust-domain/path. The SPIFFE Verifiable Identity Document (SVID) is a credential carrying the SPIFFE ID, available in two forms: an X.509-SVID (a standard X.509 certificate with the SPIFFE ID in the Subject Alternative Name field) and a JWT-SVID (a JWT assertion carrying the SPIFFE ID).

SPIRE implements the SPIFFE specification through a server-agent architecture. SPIRE agents run alongside workloads and obtain SVIDs from the SPIRE server via the SPIFFE Workload API. The SPIRE server issues short-lived SVIDs (typically with 1-hour validity) and automatically renews them before expiry, providing a form of autonomous rotation within a single trust domain.

SPIFFE's design insights are directly relevant to AGEX and have influenced it. The insight that workload identity should be infrastructure-managed rather than operator-managed is foundational to both protocols. The use of short-lived credentials with automatic renewal (SPIFFE's analogue of ARP) addresses the credential staleness problem within the SPIFFE operational scope. The attestation-based identity model, where workload identity is derived from verifiable attributes of the workload's execution environment rather than from pre-shared secrets, is a security improvement over static client credentials.

However, SPIFFE has four significant limitations in the context of autonomous agent credential management. SPIFFE is designed for workload-to-workload authentication within a single organisational trust domain. The SPIFFE federation specification exists but is complex and infrequently deployed; cross-organisational agent-to-API scenarios (the dominant use case for AI agents) are not SPIFFE's operational target. SPIFFE SVIDs carry identity assertions but no semantic information about what the workload intends to do with access; there is no analogue to the Intent Manifest. SPIFFE provides no delegation primitive; one workload cannot create a constrained sub-credential for a child process. And SPIFFE provides no cross-service emergency revocation; revoking a compromised workload's access requires action at each individual service.

2.3 Verifiable Credentials and Decentralised Identity

The W3C Decentralised Identifiers (DID) specification [W3C-DID] defines a new type of globally unique identifier that is under the control of the identifier's subject without requiring a centralised registry. DIDs resolve to DID Documents, which contain public key material and service endpoint references. The W3C Verifiable Credentials (VC) specification [W3C-VC] defines a standard for cryptographically verifiable claims about a subject, issued by an issuer and presented to a verifier.

AGEX's Agent Identity Document draws directly on DID and VC concepts. The AID structure is compatible with VC representation, and AID identifiers may optionally be expressed as DIDs to support interoperability with the broader decentralised identity ecosystem. However, AGEX does not require DID infrastructure: the AGEX Hub provides a simpler centralised identity registry that is operationally more accessible for the target deployment context.

The key distinction between the VC/DID approach and AGEX is operational scope. VCs and DIDs are designed for identity assertion and presentation scenarios, where a subject presents credentials to a verifier to establish their identity or qualifications. AGEX extends beyond identity assertion to define the full operational lifecycle of the access credentials that result from successful identity verification. The AID is the identity primitive; the CLC is the access primitive; and the ARP, delegation chains, and ERS are the lifecycle management primitives. The VC/DID ecosystem provides the former but not the latter three.

3. Threat Model and Security Requirements

3.1 System Participants

The AGEX threat model identifies the following system participants and their trust assumptions.

An Agent is an autonomous software process that requests and uses credentials to access protected resources on behalf of a human principal or organisation. Agents are assumed to be potentially compromised at any point in their operational lifecycle. The threat model does not assume that agents are trustworthy; rather, the protocol is designed so that compromised agents cause bounded harm that is detectable, containable, and recoverable.

A Service Provider is an organisation that exposes protected resources via an API and issues credentials to authorised agents under CLC terms. Service providers are assumed to be honest and to correctly implement the AGEX protocol. A service provider that issues credentials in violation of CLC terms is outside the threat model; the governance framework addresses this through the certification and audit mechanisms described in Section 9.

The AGEX Hub is the infrastructure component that mediates identity verification and credential brokering between agents and service providers. The Hub is modelled as honest-but-curious: it correctly executes the protocol but may attempt to learn information beyond what it needs to perform its function. The Hub's zero-knowledge design (Section 4.3, credential_envelope) is the primary mechanism addressing this threat. Hub operators are subject to binding terms of service enforced through the certification programme (Section 9.3).

A Human Principal is the human or organisation on whose behalf an agent operates. Human principals are assumed to be honest. They define authorisation policies through the AGEX Policy Language and are the ultimate authority for emergency revocation. Human principals are not present in real time during agent operation; their authority is mediated through pre-configured policies and escalation thresholds.

An Identity Authority is an organisation certified by the AGEX Foundation to issue AIDs. Identity Authorities are assumed to be honest within their certification terms. A compromised Identity Authority is a significant threat addressed in Section 7.5 (Attack Analysis, Attack 2).

An Adversary is a threat actor attempting to gain unauthorised access to protected resources, disrupt agent operation, extract credential material, or undermine the integrity of the audit record. We model adversaries at two capability levels, described in Section 3.2.

3.2 Adversary Capabilities

We consider two adversary capability levels.

A standard network adversary can observe, intercept, modify, and replay network traffic between any pair of protocol participants on any network path. The standard network adversary cannot break standard cryptographic primitives in polynomial time. Specifically, we assume: the hardness of the Computational Diffie-Hellman (CDH) problem on Curve25519 [Bernstein06], the unforgeability of Ed25519 signatures under chosen-message attack (EUF-CMA) [RFC8032], and the IND-CCA2 security of AES-256-GCM with random 96-bit nonces. We further assume that the HKDF key derivation function is a pseudorandom function family (PRF). All of these assumptions are standard in the cryptographic literature and are satisfied by the named primitives under the best current understanding.

An advanced persistent threat (APT) adversary additionally has the ability to compromise agent processes after deployment, obtaining full control of the agent's private key material, memory contents, environment variables, and credential store. The APT adversary cannot simultaneously compromise the Hub and a service provider; we assume that compromising multiple independent organisations requires resources beyond those of the adversary being modelled.

We explicitly exclude from scope the following threat scenarios: physical compromise of Hub data centre infrastructure; side-channel attacks on cryptographic implementations; compromise of the human principal's policy management interface via social engineering; attacks on the physical infrastructure of service providers; and global cryptographic breaks of the primitive suite. These exclusions are standard in protocol security analysis and do not reflect an assumption that such attacks are impossible, only that addressing them is outside the scope of a protocol specification.

3.3 Security Requirements

We define eight security requirements that AGEX must satisfy. These requirements are derived from the operational needs identified in Section 1 and the threat model established in Sections 3.1 and 3.2.

SR-1: Credential Confidentiality. Credential material (API keys, tokens, secrets) issued under a CLC MUST NOT be accessible to any party other than the issuing service provider and the beneficiary agent identified in the CLC. This requirement applies to the AGEX Hub, to network adversaries, and to other agents holding valid credentials from the same service provider.

SR-2: Agent Authentication. A service provider MUST be able to verify the identity of any agent requesting credentials with cryptographic assurance tied to an AID issued by a certified Identity Authority. An adversary who does not hold the private key corresponding to a valid AID MUST NOT be able to obtain credentials as that agent.

SR-3: Forward Secrecy Under Rotation. Compromise of a credential issued at time T MUST NOT enable recovery of credentials issued in rotation cycles before time T, nor of credentials issued after the next successful rotation event following time T. Each rotation cycle MUST produce cryptographically independent credentials whose security is not affected by the compromise of any other rotation cycle's credential material.

SR-4: Delegation Scope Monotonicity. A sub-credential issued via the delegation chain mechanism MUST carry a scope that is a strict subset of the delegating agent's current granted_scopes at the time of delegation. It MUST be cryptographically impossible for a delegated credential to carry scopes exceeding those of its parent credential.

SR-5: Cascade Revocation Completeness. A revocation event initiated against any agent at any node in a delegation chain MUST propagate to all descendant nodes within a defined completion bound of 60 seconds, leaving no valid sub-credentials outstanding beyond that bound. The completion of cascade revocation MUST be auditable.

SR-6: Intent Integrity. The Intent Manifest submitted by an agent at credential request time MUST be cryptographically bound to the issued CLC such that post-hoc modification of declared intent is detectable by any party with access to the manifest and the CLC. Modifications to intent declarations after CLC issuance MUST be detectable.

SR-7: Audit Non-Repudiation. All credential lifecycle events (issuance, rotation, escalation, delegation, revocation) MUST be recorded in a tamper-evident audit log such that an agent cannot credibly deny having requested, held, or used a credential. The audit log MUST be structured and machine-queryable.

SR-8: Hub Zero-Knowledge. The AGEX Hub MUST NOT be able to decrypt credential material in transit or at rest, even with full read access to its own database, key material, and network traffic. An insider at the Hub MUST NOT be able to obtain usable credential material for any agent.

4. Protocol Primitives and Schemas

AGEX introduces five protocol primitives. Each primitive is fully specified with a normative JSON schema, field-level descriptions, and implementation requirements. All schemas use JSON Schema conventions with normative annotations.

4.1 Agent Identity Document (AID)

The Agent Identity Document is the foundational identity primitive of AGEX. It is a signed data structure establishing the verifiable identity of an agent within the AGEX trust infrastructure. AIDs are issued by AGEX-registered Identity Authorities.

`AID Schema v1.0`

```
Agent Identity Document (AID) - Full Schema v1.0

{
  "aid_version":    "1.0",
  "aid_id":         "<uuid-v4>",
  "issuer": {
    "ia_id":        "<uuid-v4>",
    "ia_did":       "did:agex:<ia_id>",
    "ia_name":      "<string>",
    "ia_cert_id":   "<uuid-v4>"
  },
  "issued_at":      "<iso8601-utc>",
  "expires_at":     "<iso8601-utc>",
  "agent": {
    "name":         "<string | null>",
    "version":      "<semver | null>",
    "framework":    "<string | null>",
    "type":         "<orchestrator|worker|specialist|gateway>",
    "capabilities": ["<string>"],
    "principal": {
      "organisation": "<string>",
      "org_id":       "<uuid-v4>",
      "contact":      "<email>",
      "jurisdiction": "<iso3166-2>"
    }
  },
  "trust_tier":     <0|1|2|3>,
  "public_key": {
    "kty":          "OKP",
    "crv":          "Ed25519",
    "x":            "<base64url-32-bytes>"
  },
  "renewal_policy": {
    "auto_renew":             <bool>,
    "renewal_window_days":    <int>,
    "renewal_requires_new_key": <bool>
  },
  "restrictions": {
    "allowed_services":       ["<service_id>"] ,
    "denied_services":        ["<service_id>"],
    "max_clc_duration_seconds": <int>,
    "max_delegation_depth":   <int>,
    "geo_restrictions":       ["<iso3166-2>"]
  },
  "ia_signature":   "<base64url-ed25519-sig>"
}
```

Field Descriptions (normative):

```
aid_version: MUST be the string "1.0" for implementations conforming to this
specification. Future versions will increment this field.
```

aid_id: MUST be a randomly generated UUID version 4 per RFC 4122 [RFC4122]. MUST be globally unique. MUST NOT be reused across AID renewals; each renewal produces a new AID with a new aid_id.

issuer.ia_id: MUST be the UUID identifier of the issuing Identity Authority as registered with the AGEX Foundation Root of Trust.

issuer.ia_did: MUST be the DID representation of the IA identifier, formed as did:agex: followed by the ia_id value. This field enables interoperability with W3C DID-based verification tooling.

issuer.ia_cert_id: MUST be the identifier of the IA certificate under which this AID was issued, enabling revocation checking against the IA's certificate validity.

issued_at: MUST be a UTC timestamp in ISO 8601 format. MUST accurately reflect the actual time of AID issuance. Implementations MUST reject AIDs where issued_at is more than 300 seconds in the future relative to the verifier's current time, to resist clock skew-based attacks.

expires_at: MUST be after issued_at. MUST NOT be more than 2 years after issued_at for Tier 0 and Tier 1 agents. MAY be up to 5 years after issued_at for Tier 2 and Tier 3 agents with documented renewal procedures. Implementations MUST reject expired AIDs.

agent.type: MUST be one of the four enumerated values. An orchestrator agent coordinates other agents and typically holds broader scope. A worker agent executes specific tasks and typically holds narrower scope. A specialist agent has domain-specific capabilities. A gateway agent mediates between agent systems and external services.

trust_tier: MUST be set by the issuing IA based on verified organisational identity and, for Tiers 2 and 3, demonstrated operational history. MUST NOT be self-asserted by the agent. The trust tier determines the level of automated approval the agent receives from Hub policy evaluation (Section 5.1.3).

public_key: MUST be an Ed25519 public key encoded in JWK format per RFC 8037 [RFC8037]. The corresponding private key MUST be held exclusively by the agent and MUST NOT be transmitted to any other party including the Hub or the issuing IA.

restrictions: OPTIONAL. When present, the Hub MUST enforce these restrictions during credential request processing and MUST reject requests that violate them.

ia_signature: MUST be an Ed25519 signature computed over the RFC 8785 [RFC8785] canonical JSON serialisation of all other AID fields. The signature MUST be verifiable using the IA's registered public key. Implementations MUST verify this signature before accepting an AID in any protocol flow.

AID Revocation: An IA MAY revoke an AID at any time by publishing a signed Revocation Record to the AGEX Hub. A Revocation Record MUST contain: the aid_id being revoked, the ia_id of the revoking IA, a revocation_reason (one of: compromise, policy_violation, agent_decommission, ia_request, principal_request), the revocation timestamp, and an ia_signature over the canonical JSON of these fields. The Hub MUST update its revocation index within 60 seconds of receiving a valid Revocation Record. Implementations checking AID validity MUST check the revocation index in addition to the expiry timestamp.

4.2 Intent Manifest

The Intent Manifest is submitted by an agent when requesting credentials. It provides a structured semantic declaration of what the agent intends to do, for how long, and under what constraints. Unlike OAuth 2.0 scopes which enumerate permissions, an Intent Manifest describes purpose.

Intent Manifest Schema v1.0

```
Intent Manifest - Full Schema v1.0

{
  "manifest_version": "1.0",
  "manifest_id":      "<uuid-v4>",
  "created_at":       "<iso8601-utc>",
  "requesting_aid":   "<aid_id>",
  "target": {
    "service_id":         "<string>",
    "service_version":    "<string | null>",
    "resource_paths":     ["<uri-pattern>"],
    "requested_scopes":   ["<string>"],
    "minimum_scopes":     ["<string>"],
    "environment":        "<production|staging|development>"
  },
  "intent": {
    "summary":            "<string-max-500>",
    "task_id":            "<uuid-v4 | null>",
    "task_type":          "<read|write|read_write|admin|transact|notify>",
    "data_classification": "<public|internal|confidential|restricted>",
    "automated":          <bool>,
    "reversible":         <bool>,
    "human_visible":      <bool>,
    "estimated_api_calls": <int | null>
  },
  "duration": {
    "estimated_start":         "<iso8601-utc>",
    "estimated_end":           "<iso8601-utc>",
    "max_duration_seconds":    <int>,
    "idle_timeout_seconds":    <int | null>,
    "allow_extension":         <bool>
  },
  "data_handling": {
    "retention_policy":        "<no_retention|session|persistent>",
    "pii_processing":          <bool>,
    "pii_categories":          ["<string>"] ,
    "cross_border_transfer":   <bool>,
    "transfer_jurisdictions":  ["<iso3166-2>"],
    "encryption_at_rest":      <bool>,
    "deletion_on_completion":  <bool>
  },
  "delegation": {
      "sub_agents_permitted":         <bool>,
      "max_delegation_depth":         <int>,
      "permitted_sub_agent_types":    ["<agent_type>"],
      "sub_agent_scope_ceiling":      ["<string>"]
    },
    "escalation": {
      "human_approval_thresholds": {
        "transaction_value_usd":    <float | null>,
        "data_records_accessed":    <int | null>,
        "new_resource_types":       <bool>,
        "geographic_expansion":     <bool>,
        "pii_volume_records":       <int | null>
      },
      "escalation_contact":        "<email>",
      "escalation_timeout_seconds": <int>,
      "timeout_action":            "<approve|deny|suspend>"
    },
    "audit": {
      "structured_logging":   <bool>,
      "log_endpoint":         "<uri | null>",
      "reporting_interval":   <int>
    },
    "agent_signature": "<base64url-ed25519-sig>"
  }
```

4.3 Credential Lifecycle Contract (CLC)

A CLC is the living agreement governing the credential relationship between a service provider and an agent. It contains the encrypted credential envelope, lifecycle policy, rotation schedule, delegation policy, and termination conditions. The CLC replaces the static API key as the primary credential primitive in AGEX-compliant systems.

CLC Schema v1.0

```
Credential Lifecycle Contract (CLC) - Full Schema v1.0

{
  "clc_version":         "1.0",
  "clc_id":              "<uuid-v4>",
  "issued_at":           "<iso8601-utc>",
  "issuer_service_id":   "<string>",
  "beneficiary_aid":     "<aid_id>",
  "manifest_id":         "<uuid-v4>",
  "manifest_hash":       "<sha3-256-hex>",
  "credential_envelope": {
    "algorithm":         "ECDH-ES+AES256GCM",
    "epk": {
      "kty":             "OKP",
      "crv":             "X25519",
      "x":               "<base64url-32-bytes>"
    },
    "ciphertext":        "<base64url>",
    "nonce":             "<base64url-12-bytes>",
    "tag":               "<base64url-16-bytes>",
    "protected":         "<base64url-json-header>"
  },
  "granted_scopes":      ["<string>"],
  "scope_ceiling":       ["<string>"],
  "denied_scopes":       ["<string>"],
  "validity": {
    "not_before":              "<iso8601-utc>",
    "not_after":               "<iso8601-utc>",
    "idle_timeout_seconds":    <int | null>,
    "geographic_restriction":  ["<iso3166-2>"]
  },
  "rotation_policy": {
    "rotation_interval_seconds":        <int>,
    "rotation_overlap_seconds":         <int>,
    "max_rotations":                    <int>,
    "rotation_requires_new_manifest":   <bool>,
    "rotation_requires_human_approval": <bool>,
    "key_derivation_function":          "HKDF-SHA256"
  },
  "delegation_policy": {
    "delegation_permitted":       <bool>,
    "max_delegation_depth":       <int>,
      "scope_reduction_required":   <bool>,
      "permitted_sub_agent_types":  ["<agent_type>"],
      "max_concurrent_sub_clcs":    <int>
    },
    "audit_requirements": {
      "reporting_interval_seconds": <int>,
      "log_endpoint":               "<uri | null>",
      "anomaly_thresholds": {
        "requests_per_minute":   <int>,
        "error_rate_percent":    <float>,
        "unusual_hours":         <bool>
      },
      "anomaly_action":           "<alert|suspend|revoke>"
    },
    "termination_conditions": [
      {
        "condition_type": "<expiry|idle_timeout|anomaly|escalation_breach|revocation_signal|parent_revoked>",
        "action":         "<revoke|suspend|alert>",
        "grace_seconds":  <int>
      }
    ],
    "chain_provenance":  ["<clc_id>"],
    "hub_binding": {
      "hub_id":        "<string>",
      "hub_timestamp": "<iso8601-utc>",
      "hub_signature": "<base64url-ed25519-sig>"
    },
    "provider_signature": "<base64url-ed25519-sig>"
  }
```

4.4 Autonomous Rotation Protocol (ARP) Messages

ARP defines four message types exchanged during a rotation event: the agent's rotation request, the Hub's forwarding message to the service provider, the service provider's response, and the Hub's delivery to the agent. Each message is signed by its originator.

ARP Message Schemas v1.0

```
ARP Message Schemas v1.0

-- Rotation Request (Agent to Hub) --
{
  "arp_version":      "1.0",
  "message_type":     "rotation_request",
  "request_id":       "<uuid-v4>",
  "clc_id":           "<uuid-v4>",
  "requesting_aid":   "<aid_id>",
  "rotation_reason":  "<scheduled|compromise_suspected|policy_change|principal_request>",
  "new_public_key": {
    "kty":            "OKP",
    "crv":            "Ed25519",
    "x":              "<base64url-32-bytes>"
  },
  "requested_validity_seconds": <int>,
  "nonce":            "<base64url-16-bytes>",
  "timestamp":        "<iso8601-utc>",
  "agent_signature":  "<base64url-ed25519-sig>"
}

-- Rotation Forwarding (Hub to Service Provider) --
{
  "message_type":     "rotation_forwarding",
  "request_id":       "<uuid-v4>",
  "clc_id":           "<uuid-v4>",
  "new_public_key":   { ... },
  "rotation_reason":  "<string>",
  "hub_timestamp":    "<iso8601-utc>",
  "hub_nonce":        "<base64url-16-bytes>",
  "hub_signature":    "<base64url-ed25519-sig>"
}

-- Rotation Response (Service Provider to Hub) --
{
  "message_type":          "rotation_response",
  "request_id":            "<uuid-v4>",
  "new_clc_id":            "<uuid-v4>",
  "new_credential_envelope": { ... },
  "old_clc_expiry":        "<iso8601-utc>",
  "overlap_seconds":       <int>,
  "provider_timestamp":    "<iso8601-utc>",
  "provider_signature":    "<base64url-ed25519-sig>"
}

-- Rotation Delivery (Hub to Agent) --
{
  "message_type":          "rotation_delivery",
  "request_id":            "<uuid-v4>",
  "new_clc":               { ... full CLC object ... },
  "old_clc_expiry":        "<iso8601-utc>",
  "hub_timestamp":         "<iso8601-utc>",
  "hub_signature":         "<base64url-ed25519-sig>"
}
```

4.5 Delegation Chain Messages

Delegation chain messages define the request, the resulting sub-CLC structure including the chain_provenance field, and the revocation signal. The chain_provenance field is the key mechanism enabling cascade revocation across arbitrarily deep delegation chains.

Delegation Chain Schemas v1.0

```
Delegation Chain Schemas v1.0

-- Delegation Request (Parent Agent to Hub) --
{
  "delegation_version":    "1.0",
  "message_type":          "delegation_request",
  "delegation_id":         "<uuid-v4>",
  "parent_clc_id":         "<uuid-v4>",
  "parent_aid":            "<aid_id>",
  "child_aid":             "<aid_id>",
  "requested_sub_scopes":  ["<string>"],
  "delegation_duration_seconds": <int>,
  "chain_depth":           <int>,
  "max_further_delegation_depth": <int>,
  "purpose":               "<string-max-500>",
  "sub_escalation": {
    "inherit_parent_thresholds": <bool>,
    "escalation_contact":        "<email | null>"
  },
  "timestamp":             "<iso8601-utc>",
  "parent_signature":      "<base64url-ed25519-sig>"
}

-- Sub-CLC (Issued by Hub on behalf of Service Provider) --
{
  ... standard CLC fields ...
  "chain_provenance": ["<parent_clc_id>", "<grandparent_clc_id>", ...],
  "delegation_id":    "<uuid-v4>",
  "chain_depth":      <int>,
  "parent_aid":       "<aid_id>"
}

-- Delegation Revocation (Any authorised party to Hub) --
{
  "message_type":       "delegation_revocation",
  "revocation_id":      "<uuid-v4>",
  "target_clc_id":      "<uuid-v4>",
  "cascade":            <bool>,
  "revocation_reason":  "<string>",
  "initiated_by_aid":   "<aid_id>",
  "timestamp":          "<iso8601-utc>",
  "initiator_signature": "<base64url-ed25519-sig>"

}
```

5. HTTP API Specification

The AGEX Hub exposes a RESTful HTTP API over TLS 1.3 [RFC8446]. All endpoints use JSON request and response bodies. All timestamps are ISO 8601 UTC. All JSON bodies are UTF-8 encoded per RFC 8259 [RFC8259]. All identifiers are UUID v4 unless otherwise specified. All signatures are Ed25519 over RFC 8785 canonical JSON, base64url encoded. The base URL for the Hub API is https:///agex/v1.

All requests MUST include the following HTTP headers:

Content-Type: application/json X-AGEX-Version: 1.0 X-AGEX-Request-ID: X-AGEX-Timestamp: X-AGEX-AID: X-AGEX-Signature:

The X-AGEX-Signature MUST be computed over the RFC 8785 [RFC8785] canonical JSON of the request body concatenated with the value of X-AGEX-Timestamp and X-AGEX-Request-ID, using the private key corresponding to the AID identified in X-AGEX-AID.

Implementations MUST reject requests where X-AGEX-Timestamp is more than 300 seconds from the Hub's current time. Implementations MUST reject duplicate X-AGEX-Request-IDs within a 24-hour deduplication window.

All error responses use the following standard error schema:

```
{
  "error": {
    "code":    "<string>",
    "message": "<string>",
    "details": { ... },
    "request_id": "<uuid-v4>",
    "timestamp":  "<iso8601-utc>"
  }
}
```

5.x POST /credentials/request -- Request Credentials

Submit a credential request for a target service. The agent submits its AID and a signed Intent Manifest. The Hub validates the AID, evaluates approval policy, and either immediately issues a CLC, routes for human approval, or rejects the request.

This is the primary credential acquisition endpoint and implements the flow described in Section 6.1.

Request

```
POST /agex/v1/credentials/request
Content-Type: application/json
X-AGEX-Version: 1.0
X-AGEX-Request-ID: 7f3a1b2c-...
X-AGEX-Timestamp: 2026-02-18T10:00:00Z
X-AGEX-AID: aid-uuid-here
X-AGEX-Signature: <sig>

{
  "aid":      { ... full AID object ... },
  "manifest": { ... full Intent Manifest object ... }
}
```

Response: 200 Approved

```
{
  "status":        "approved",
  "clc":           { ... full CLC object ... },
  "request_id":    "<uuid-v4>",
  "hub_timestamp": "<iso8601-utc>"
}
```

Response: 202 Pending

```
{
  "status":           "pending_human_approval",
  "approval_token":   "<uuid-v4>",
  "estimated_review_seconds": <int>,
  "poll_endpoint":    "/agex/v1/credentials/approval/<approval_token>",
  "request_id":       "<uuid-v4>"
}
```

Response: 400 Bad Request

```
{
  "error": {
    "code":    "INVALID_MANIFEST",
    "message": "Intent Manifest failed schema validation",
    "details": { "field": "intent.task_type", "reason": "Unknown value" },
    "request_id": "<uuid-v4>"
  }
}
```

Response: 401 Unauthorized

```
{
  "error": {
    "code":    "AID_INVALID",
    "message": "AID signature verification failed or AID is revoked",
    "request_id": "<uuid-v4>"
  }
}
```

Response: 403 Forbidden

```
{
  "error": {
    "code":    "REQUEST_REJECTED",
    "message": "Credential request rejected by policy engine",
    "details": { "policy_rule": "tier0_admin_denied", "scope": "admin:write" },
    "request_id": "<uuid-v4>"
  }
}
```

Error Codes

Error Code	Description
INVALID_MANIFEST	Manifest failed JSON schema validation
AID_EXPIRED	AID validity period has elapsed
AID_REVOKED	AID has been revoked by the issuing IA
AID_SIGNATURE_INVALID	ia_signature on AID failed verification
MANIFEST_SIGNATURE_INVALID	agent_signature on manifest failed verification
SERVICE_NOT_FOUND	target.service_id is not registered with this Hub
SCOPE_NOT_AVAILABLE	One or more requested_scopes are not offered by the service
REQUEST_REJECTED	Policy engine rejected the request
TIMESTAMP_OUT_OF_RANGE	Request timestamp more than 300 seconds from Hub time

Error Code	Description
DUPLICATE_REQUEST_ID	X-AGEX-Request-ID has been seen within the deduplication window

5.x GET /credentials/approval/{approval_token} -- Poll Approval Status

Poll the status of a pending human approval request. Returns the current status and, if approved, the issued CLC. Clients SHOULD poll at intervals of no less than 30 seconds. The Hub MAY return a Retry-After header indicating the recommended poll interval.

Request

```
GET /agex/v1/credentials/approval/7f3a1b2c-...
X-AGEX-Version: 1.0
X-AGEX-Request-ID: <new-uuid>
X-AGEX-Timestamp: 2026-02-18T10:05:00Z
X-AGEX-AID: <aid_id>
X-AGEX-Signature: <sig-over-empty-body>
```

Response: 200 Pending

```
{
  "status":          "pending",
  "approval_token":  "<uuid-v4>",
  "created_at":      "<iso8601-utc>",
  "timeout_at":      "<iso8601-utc>"
}
```

Response: 200 Approved

```
{
  "status":     "approved",
  "clc":        { ... full CLC object ... },
  "approved_by": "<email>",
  "approved_at": "<iso8601-utc>"
}
```

Response: 200 Rejected

```
{
  "status":      "rejected",
  "rejected_by": "<email>",
  "rejected_at": "<iso8601-utc>",
  "reason":      "<string>"
}
```

Response: 200 Timed Out

```
{
  "status":      "timed_out",
  "timeout_at":  "<iso8601-utc>",
  "action_taken": "<approve|deny|suspend>"
}
```

Error Codes

Error Code	Description
APPROVAL_NOT_FOUND	approval_token not found or expired
APPROVAL_NOT_OWNER	Requesting AID is not the AID that created this approval request

5.x POST /credentials/rotate -- Rotate Credentials (ARP)

Initiate the Autonomous Rotation Protocol for an active CLC. The agent submits a new public key; the Hub coordinates with the service provider to issue a new CLC encrypted under the new key, with an atomic overlap window ensuring no access gap.

This endpoint implements the ARP flow described in Section 6.2.

Request

```
POST /agex/v1/credentials/rotate
Content-Type: application/json
X-AGEX-Version: 1.0
X-AGEX-Request-ID: <uuid-v4>
X-AGEX-Timestamp: <iso8601-utc>
X-AGEX-AID: <aid_id>
X-AGEX-Signature: <sig>

{
  "arp_version":    "1.0",
  "message_type":   "rotation_request",
  "request_id":     "<uuid-v4>",
  "clc_id":         "<uuid-v4>",
  "requesting_aid": "<aid_id>",
  "rotation_reason": "scheduled",
  "new_public_key": {
    "kty": "OKP", "crv": "Ed25519", "x": "<base64url>"
  },
  "requested_validity_seconds": 86400,
  "nonce":         "<base64url-16-bytes>",
  "timestamp":     "<iso8601-utc>",
  "agent_signature": "<sig>"
}
```

Response: 200 Success

```
{
  "status":          "rotated",
  "new_clc":         { ... full CLC object ... },
  "old_clc_expiry":  "<iso8601-utc>",
  "overlap_seconds": <int>,
  "hub_timestamp":   "<iso8601-utc>"
}
```

Response: 409 Conflict

```
{
  "error": {
    "code":    "ROTATION_IN_PROGRESS",
    "message": "A rotation for this CLC is already in progress",
    "existing_request_id": "<uuid-v4>"
  }
}
```

Error Codes

Error Code	Description
CLC_NOT_FOUND	clc_id not found for this agent
CLC_EXPIRED	CLC has already expired; rotation not possible

Error Code	Description
CLC_REVOKED	CLC has been revoked; rotation not possible
ROTATION_IN_PROGRESS	Concurrent rotation request already active for this CLC
KEY_REUSE	new_public_key matches current or recent key material
SP_UNAVAILABLE	Service provider did not respond within the rotation timeout

5.x POST /credentials/escalate -- Request Scope Escalation

Request elevated permissions for an active CLC. The agent specifies the additional scopes required and a justification. The Hub evaluates the request against the CLC's scope_ceiling and the agent's trust tier, then either auto-approves or routes for human review.

Request

```
POST /agex/v1/credentials/escalate
Content-Type: application/json
...headers...

{
  "clc_id":              "<uuid-v4>",
  "requesting_aid":      "<aid_id>",
  "additional_scopes":   ["<string>"],
  "justification":       "<string-max-1000>",
  "task_id":             "<uuid-v4 | null>",
  "estimated_duration_seconds": <int>,
  "timestamp":           "<iso8601-utc>",
  "agent_signature":     "<sig>"
}
```

Response: 200 Approved

```
{
  "status":          "approved",
  "updated_clc":     { ... CLC with new granted_scopes ... },
  "hub_timestamp":   "<iso8601-utc>"
}
```

Error Codes

Error Code	Description
SCOPE_CEILING_EXCEEDED	Requested scopes exceed the CLC scope_ceiling
ESCALATION_DENIED	Policy engine denied the escalation request
MAX_ESCALATIONS_REACHED	CLC has reached maximum number of escalation events

5.x POST /credentials/delegate -- Create Delegation

Create a sub-CLC for a child agent. The parent agent specifies the child AID, the sub-scopes to delegate (MUST be a strict subset of parent CLC's granted_scopes), and the delegation duration. The Hub validates the request and issues a sub-CLC.

Request

```
POST /agex/v1/credentials/delegate
Content-Type: application/json
...headers...

{
  "delegation_version":   "1.0",
  "message_type":         "delegation_request",
  "delegation_id":        "<uuid-v4>",
  "parent_clc_id":        "<uuid-v4>",
  "parent_aid":           "<aid_id>",
  "child_aid":            "<aid_id>",
  "requested_sub_scopes": ["<string>"],
  "delegation_duration_seconds": <int>,
  "chain_depth":          <int>,
  "max_further_delegation_depth": <int>,
  "purpose":              "<string>",
  "timestamp":            "<iso8601-utc>",
  "parent_signature":     "<sig>"
}
```

Response: 200 Success

```
{
  "status":     "delegated",
  "sub_clc":    { ... CLC with chain_provenance populated ... },
  "delegation_id": "<uuid-v4>"
}
```

Error Codes

Error Code	Description
SCOPE_CEILING_EXCEEDED	Requested sub-scopes not subset of parent granted_scopes
MAX_DEPTH_EXCEEDED	chain_depth exceeds parent CLC max_delegation_depth
CHILD_AID_INVALID	child_aid is expired, revoked, or not found
DELEGATION_NOT_PERMITTED	Parent CLC delegation_policy.delegation_permitted is false
MAX_SUB_CLCS_EXCEEDED	Parent CLC max_concurrent_sub_clcs limit reached

5.x POST /ers/signal -- Emergency Revocation System

Initiate emergency revocation of all credentials held by a target agent. This endpoint is available to the agent's owning organisation, any service provider that has issued a CLC to the agent, the issuing Hub operator, and the issuing IA. Cascade revocation propagates to all sub-CLCs in delegation chains rooted at the target AID within 60 seconds.

Request

```
POST /agex/v1/ers/signal
Content-Type: application/json
...headers...

{
  "message_type":    "ers_signal",
  "signal_id":       "<uuid-v4>",
  "target_aid":      "<aid_id>",
  "reason":          "<compromise|policy_violation|decommission|legal_hold|test>",
  "initiated_by":    "<aid_id | org_id | hub_id>",
  "initiator_type":  "<agent|organisation|hub|ia>",
  "cascade":         true,
  "timestamp":       "<iso8601-utc>",
  "initiator_signature": "<sig>"
}
```

Response: 200 Success

```
{
  "status":             "revocation_initiated",
  "signal_id":          "<uuid-v4>",
  "clcs_revoked":       <int>,
  "sub_clcs_revoked":   <int>,
  "services_notified":  <int>,
  "completion_time_ms": <int>,
  "audit_record_id":    "<uuid-v4>"
}
```

Error Codes

Error Code	Description
INITIATOR_NOT_AUTHORISED	Initiating entity is not authorised to revoke target AID
TARGET_AID_NOT_FOUND	target_aid not found in Hub registry
ERS_TIMEOUT	Cascade revocation did not complete within 60-second bound

5.x GET /audit/events -- Query Audit Log

Query the structured audit log for credential lifecycle events. Results are paginated. Callers may only query events related to their own AID or, for organisations, events related to any AID issued under their org_id.

Request

```
GET /agex/v1/audit/events
  ?aid=<aid_id>
  &clc_id=<uuid-v4>
  &event_type=<issuance|rotation|escalation|delegation|revocation|ers>
  &from=<iso8601-utc>
  &to=<iso8601-utc>
  &page_token=<string>
  &page_size=<int-max-100>
```

Response: 200 Success

```
{
  "events": [
    {
      "event_id":    "<uuid-v4>",
      "event_type":  "<string>",
      "timestamp":   "<iso8601-utc>",
      "aid":         "<aid_id>",
      "clc_id":      "<uuid-v4 | null>",
      "service_id":  "<string | null>",
      "details":     { ... event-specific fields ... },
      "hash":        "<sha3-256-hex>",
      "prev_hash":   "<sha3-256-hex>"
    }
  ],
  "next_page_token": "<string | null>",
  "total_count": <int>
}
```

Error Codes

Error Code	Description
QUERY_UNAUTHORISED	Caller not authorised to query events for specified AID
DATE_RANGE_TOO_LARGE	Requested date range exceeds maximum of 90 days

5.x GET /discovery/services -- Service Provider Discovery

Discover AGEX-registered service providers and their capability metadata. Returns a paginated list of registered services with their supported scopes, required trust tiers, and policy endpoint references. Analogous to OAuth 2.0 server metadata (RFC 8414).

Request

```
GET /agex/v1/discovery/services
  ?capability=<string>
  &min_trust_tier=<0|1|2|3>
  &category=<string>
  &page_token=<string>
```

Response: 200 Success

```
{
  "services": [
    {
      "service_id":        "<string>",
      "name":              "<string>",
      "description":       "<string>",
      "version":           "<string>",
      "category":          "<string>",
      "available_scopes":  ["<string>"],
      "min_trust_tier":    <int>,
      "policy_endpoint":   "<uri>",
      "agex_endpoint":     "<uri>",
      "documentation_url": "<uri>"
    }
  ],
  "next_page_token": "<string | null>"
}
```

5.x POST /aids/register -- Register AID

Register a newly issued AID with the Hub. AIDs MUST be registered before they can be used in credential requests. Registration allows the Hub to cache AID metadata for efficient validation and to begin monitoring for revocation events from the issuing IA.

Request

```
POST /agex/v1/aids/register
Content-Type: application/json
...headers...

{
  "aid": { ... full AID object ... },
  "ia_id": "<uuid-v4>",
  "registration_timestamp": "<iso8601-utc>"
}
```

Response: 200 Success

```
{
  "status":             "registered",
  "aid_id":             "<uuid-v4>",
  "hub_id":             "<string>",
  "registered_at":      "<iso8601-utc>",
  "next_renewal_check": "<iso8601-utc>"
}
```

Error Codes

Error Code	Description
AID_ALREADY_REGISTERED	aid_id is already registered with this Hub
IA_NOT_TRUSTED	Issuing IA is not in this Hub's trusted IA list
AID_SIGNATURE_INVALID	ia_signature verification failed

6. AGEX Policy Language (APL)

The AGEX Policy Language (APL) is a declarative, JSON-based language [RFC8259] for expressing credential approval policies. APL policies are authored by service providers and human principals, stored by the Hub Policy Engine, and evaluated against incoming credential requests to determine whether auto-approval, human review, or rejection is appropriate.

APL is designed to be human-readable, machine-evaluable, version-controlled, and auditable. A policy document is a JSON object with a defined schema that the Hub Policy Engine interprets deterministically. Policy evaluation MUST be deterministic: given the same policy document, the same request, and the same evaluation context, the Policy Engine MUST always produce the same outcome.

APL supports the following policy constructs: trust tier gates, scope allowlists and denylists, intent classification rules, data classification rules, time-bounded policies, geographic restrictions, organisation-specific rules, and composite rules using AND, OR, and NOT operators.

6.1 Policy Document Schema

APL Policy Document Schema v1.0

```
APL Policy Document - Full Schema v1.0

{
  "apl_version":   "1.0",
  "policy_id":     "<uuid-v4>",
  "service_id":    "<string>",
  "authored_by":   "<org_id | aid_id>",
  "created_at":    "<iso8601-utc>",
  "updated_at":    "<iso8601-utc>",
  "description":   "<string>",
  "default_action": "<approve|review|reject>",
  "rules": [
    {
      "rule_id":       "<string>",
      "description":   "<string>",
      "priority":      <int>,
      "condition":     { ... APL Condition object ... },
      "action":        "<approve|review|reject>",
      "action_params": {
        "review_timeout_seconds": <int>,
        "timeout_action":        "<approve|reject>",
        "notify":                ["<email>"],
        "reason":                "<string>"
      }
    }
  ],
  "scope_policy": {
    "allowed_scopes":   ["<string>"],
    "denied_scopes":    ["<string>"],
    "scope_tiers": {
      "tier0": ["<string>"],
      "tier1": ["<string>"],
      "tier2": ["<string>"],
      "tier3": ["<string>"]
    }
  },
  "audit_policy": {
    "log_all_decisions": <bool>,
    "log_approved_only": <bool>,
    "alert_on_reject":   <bool>
  }
}
```

6.2 Condition Types

APL supports the following condition types. Conditions may be nested to arbitrary depth using the composite AND, OR, and NOT operators.

APL Condition Reference

```
APL Condition Objects

Conditions evaluate to true or false given a credential request context.
Conditions may be simple predicates or composite boolean expressions.

-- Trust Tier Condition --
{
  "type":     "trust_tier",
  "operator": "<gte|lte|eq>",
  "value":    <0|1|2|3>
}

-- Scope Condition --
{
  "type":     "scope",
  "operator": "<contains|not_contains|subset_of>",
  "value":    ["<scope_string>"]
}

-- Intent Type Condition --
{
  "type":     "intent_type",
  "operator": "<in|not_in>",
  "value":    ["<read|write|read_write|admin|transact|notify>"]
}

-- Data Classification Condition --
{
  "type":     "data_classification",
  "operator": "<in|not_in>",
  "value":    ["<public|internal|confidential|restricted>"]
}

-- Duration Condition --
{
  "type":     "duration",
  "operator": "<lte|gte>",
  "field":    "<max_duration_seconds|estimated_duration_seconds>",
  "value":    <int>
}

-- Organisation Condition --
```

```
{
  "type":     "organisation",
  "operator": "<in|not_in>",
  "value":    ["<org_id>"]
}

-- Time Window Condition --
{
  "type":       "time_window",
  "days":       ["<mon|tue|wed|thu|fri|sat|sun>"],
  "hours_utc":  { "from": "09:00", "to": "17:00" }
}

-- Geographic Condition --
{
  "type":     "geography",
  "operator": "<in|not_in>",
  "value":    ["<iso3166-2>"]
}

-- PII Condition --
{
  "type":  "pii_processing",
  "value": <bool>
}

-- Composite AND Condition --
{
  "type":       "and",
  "conditions": [ { ... }, { ... } ]
}

-- Composite OR Condition --
{
  "type":       "or",
  "conditions": [ { ... }, { ... } ]
}

-- Composite NOT Condition --
{
  "type":      "not",
  "condition": { ... }
```

```
}
```

6.3 Example Policy Document

The following example illustrates a realistic APL policy for a production payments API, demonstrating trust tier gates, time window conditions, human review routing, and data classification enforcement.

Example: ACME Payments API Policy

```
Example APL Policy: Production API Service

{
  "apl_version": "1.0",
  "policy_id": "pol-3f2a1b9c-...",
  "service_id": "acme-payments-api",
  "description": "Credential policy for ACME Payments API",
  "default_action": "reject",

  "scope_policy": {
    "allowed_scopes": [
      "payments:read", "payments:write", "refunds:read",
      "refunds:write", "customers:read", "reports:read"
    ],
    "denied_scopes": ["admin:all", "keys:manage"],
    "scope_tiers": {
      "tier0": ["payments:read", "customers:read"],
      "tier1": ["payments:read", "payments:write", "customers:read"],
      "tier2": ["payments:read", "payments:write", "customers:read",
                "refunds:read", "refunds:write", "reports:read"],
      "tier3": ["payments:read", "payments:write", "customers:read",
                "refunds:read", "refunds:write", "reports:read"]
    }
  },

  "rules": [
    {
      "rule_id": "rule-1-tier3-auto-approve",
      "description": "Tier 3 agents get automatic approval for tier-appropriate scopes",
      "priority": 10,
      "condition": {
        "type": "and",
        "conditions": [
          { "type": "trust_tier", "operator": "eq", "value": 3 },
          { "type": "scope", "operator": "subset_of",
            "value": ["payments:read","payments:write","customers:read",
                      "refunds:read","refunds:write","reports:read"] },
          { "type": "data_classification", "operator": "not_in",
            "value": ["restricted"] }
        ]
      },
      "action": "approve"
      },
      {
        "rule_id": "rule-2-tier2-business-hours",
        "description": "Tier 2 agents approved during business hours for non-transact intents",
        "priority": 20,
        "condition": {
          "type": "and",
          "conditions": [
            { "type": "trust_tier", "operator": "gte", "value": 2 },
            { "type": "intent_type", "operator": "not_in", "value": ["admin"] },
            { "type": "time_window",
              "days": ["mon","tue","wed","thu","fri"],
              "hours_utc": { "from": "08:00", "to": "18:00" } },
            { "type": "duration", "operator": "lte",
              "field": "max_duration_seconds", "value": 86400 }
          ]
        },
        "action": "approve"
      },
      {
        "rule_id": "rule-3-high-value-review",
        "description": "Route to human review for high-value transaction scope outside hours",
        "priority": 30,
        "condition": {
          "type": "and",
          "conditions": [
            { "type": "scope", "operator": "contains",
              "value": ["payments:write"] },
            { "type": "trust_tier", "operator": "lte", "value": 1 }
          ]
        },
        "action": "review",
        "action_params": {
          "review_timeout_seconds": 3600,
          "timeout_action": "reject",
          "notify": ["security@acme.example.com"],
          "reason": "Write access to payments API requires human approval for Tier 0-1 agents"
        }
      },
      {
        "rule_id": "rule-4-pii-restricted",
        "description": "Reject all requests involving restricted data classification",
        "priority": 5,
        "condition": {
          "type": "data_classification",
          "operator": "in",
          "value": ["restricted"]
        },
        "action": "reject",
        "action_params": {
          "reason": "Restricted data classification requires manual onboarding process"
        }
      }
    ]
  }
```

7. Hub Architecture and Federation Protocol (AHFP)

7.1 Hub Internal Architecture

The AGEX Hub implements six internal subsystems: Identity Registry (AID storage and revocation), Credential Broker (request routing and CLC delivery), Rotation Coordinator (ARP state machine), Policy Engine (APL evaluation), Audit Ledger (tamper-evident event log), and Emergency Revocation System. Each subsystem is independently scalable and the ERS subsystem is isolated from normal credential flows to ensure availability under high load.

Subsystem	Function	Availability Target
Identity Registry	AID validation, revocation index	99.99%
Credential Broker	Request routing, CLC delivery	99.9%
Rotation Coordinator	ARP state machine, overlap guarantee	99.99%
Policy Engine	APL evaluation for all requests	99.99%
Audit Ledger	Tamper-evident event recording	99.999%
Emergency Revocation System	Cascade revocation, 60s bound	99.999%

7.2 AHFP Cross-Hub Credential Flow

The AGEX Hub Federation Protocol (AHFP) defines how two or more AGEX Hub instances interoperate to support credential requests that cross organisational boundaries. A cross-Hub scenario arises when an agent whose AID was issued by an Identity Authority registered with Hub-A requests credentials from a service provider registered with Hub-B.

Without federation, the agent would need separate registrations at each Hub, and the service provider would need to trust each agent's Hub independently. With AHFP, Hub-A can attest the agent's identity to Hub-B, and Hub-B can verify that attestation against the AGEX Root of Trust, enabling a seamless cross-organisational credential flow.

AHFP operates on the following trust model: all participating Hubs trust the AGEX Root of Trust as the ultimate certificate authority. A Hub's certificate is issued by the Root of Trust and carries the Hub's identifier, public key, and authorised trust domains. Hubs verify each other's certificates against the Root of Trust before establishing a federation relationship. Once a federation relationship is established, Hubs can forward credential requests and attest agent identities to each other.

AHFP Cross-Hub Flow

```
AHFP Cross-Hub Credential Request Flow

Participants: Agent (A), Agent's Hub (H-A), Service Provider's Hub (H-B),
              Service Provider (SP), AGEX Root of Trust (RoT)

Pre-condition: H-A and H-B have established a federation relationship verified
               against the RoT. This is done out-of-band at Hub operator setup time.

Step 1: Agent Registration Check
  A presents AID to H-A (AID was issued by IA registered with H-A)
  H-A verifies AID signature and revocation status
  H-A determines that target service_id is registered with H-B, not H-A

Step 2: Cross-Hub Attestation Request
  H-A generates a Cross-Hub Attestation (CHA):
  {
    "cha_version":   "1.0",
    "cha_id":        "<uuid-v4>",
    "agent_aid":     "<aid_id>",
    "agent_trust_tier": <int>,
    "attesting_hub": "<hub_id_A>",
    "target_hub":    "<hub_id_B>",
    "manifest_id":   "<uuid-v4>",
    "timestamp":     "<iso8601-utc>",
    "valid_for_seconds": 300,
    "hub_a_signature": "<sig>"
  }
  H-A sends CHA and original manifest to H-B

Step 3: Cross-Hub Verification at H-B
  H-B receives CHA and manifest
  H-B verifies hub_a_signature using H-A's certified public key
  H-B verifies H-A's Hub certificate against RoT
  H-B performs independent AID revocation check against its own revocation index
  H-B evaluates SP's APL policy against the manifest and attested trust tier
  If approved, H-B forwards to SP

Step 4: CLC Issuance and Return
  SP issues CLC to H-B
  H-B wraps CLC in a Cross-Hub CLC Wrapper:
  {
    "wrapper_version": "1.0",
    "clc": { ... full CLC ... },
    "cha_id": "<uuid-v4>",
    "hub_b_id": "<hub_id_B>",
    "hub_b_timestamp": "<iso8601-utc>",
    "hub_b_signature": "<sig>"
  }
  H-B returns wrapped CLC to H-A
  H-A verifies wrapper and delivers CLC to Agent

Step 5: ARP and Lifecycle Operations
  All subsequent ARP rotation, escalation, and delegation requests
  are routed through the same H-A to H-B path.
  The CLC's hub_binding reflects H-B (the issuing Hub).
  The Agent interacts with H-A for all operations; H-A proxies to H-B transparently.
```

7.3 Hub Federation Discovery

Hub Federation Metadata Document

```
Hub Federation Discovery

Hubs publish a Federation Metadata Document at a well-known endpoint:
GET /.well-known/agex-federation

Response:
{
  "hub_id":              "<string>",
  "hub_version":         "1.0",
  "hub_public_key": {
    "kty": "OKP", "crv": "Ed25519", "x": "<base64url>"
  },
  "hub_certificate":    "<base64url-x509>",
  "trust_domains":      ["<string>"],
  "federation_endpoint": "https://<hub>/agex/v1/federation",
  "services_endpoint":   "https://<hub>/agex/v1/discovery/services",
  "root_of_trust_url":  "https://root.agex.api/certs",
  "supported_versions": ["1.0"],
  "rate_limits": {
    "federation_requests_per_minute": <int>
  }
}
```

8. Protocol Flows and Pseudocode

This section provides pseudocode for all five standard AGEX protocol flows. Pseudocode uses a Python-like syntax for readability. All cryptographic operations reference the primitive suite defined in Section 3.2. MUST and SHOULD annotations in comments indicate normative requirements.

8.1 Credential Acquisition

```
-- CREDENTIAL ACQUISITION FLOW --
-- Participants: Agent (A), Hub (H), Service Provider (SP), [Human Approver (HA)] --

function A.REQUEST_CREDENTIALS(service_id, intent_params):
    aid           = LOAD_AID_FROM_SECURE_STORAGE()
    private_key   = LOAD_PRIVATE_KEY_FROM_HSM()
    VERIFY(aid.expires_at > NOW() + 300, "AID must not expire during request window")
    VERIFY(NOT AID_IS_REVOKED(aid.aid_id), "AID must not be revoked")

    manifest = {
        manifest_version: "1.0",
        manifest_id:       UUID4(),
        created_at:        NOW(),
        requesting_aid:    aid.aid_id,
        target: {
            service_id:        service_id,
            resource_paths:    intent_params.resource_paths,
            requested_scopes:  intent_params.scopes,
            minimum_scopes:    intent_params.min_scopes,
            environment:       intent_params.environment
        },
        intent: {
            summary:           intent_params.summary,
            task_type:         intent_params.task_type,
            data_classification: intent_params.data_class,
            automated:         true,
            reversible:        intent_params.reversible,
            human_visible:     intent_params.human_visible
        },
        duration: {
            estimated_start:       NOW(),
            estimated_end:         NOW() + intent_params.duration_seconds,
            max_duration_seconds:  intent_params.duration_seconds,
            idle_timeout_seconds:  intent_params.idle_timeout
        },
        data_handling:   intent_params.data_handling,
        delegation:      intent_params.delegation_policy,
        escalation:      intent_params.escalation_thresholds
    }
    manifest.agent_signature = ED25519_SIGN(CANONICAL_JSON(manifest), private_key)

    request_id = UUID4()
    timestamp  = NOW()
    body       = {aid: aid, manifest: manifest}
    sig        = ED25519_SIGN(CANONICAL_JSON(body) + timestamp + request_id, private_key)

    response = HTTP_POST(
        url      = HUB_URL + "/agex/v1/credentials/request",
        headers  = {X-AGEX-Version: "1.0", X-AGEX-Request-ID: request_id,
                    X-AGEX-Timestamp: timestamp, X-AGEX-AID: aid.aid_id,
                    X-AGEX-Signature: sig},
        body     = body
    )
    RETURN A.HANDLE_CREDENTIAL_RESPONSE(response, aid, private_key)

function A.HANDLE_CREDENTIAL_RESPONSE(response, aid, private_key):
    IF response.status == 200 AND response.body.status == "approved":
        clc = response.body.clc
        VERIFY_ED25519(clc.provider_signature, SP_PUBLIC_KEY, CANONICAL_JSON(clc))
        VERIFY_ED25519(clc.hub_binding.hub_signature, HUB_PUBLIC_KEY, CANONICAL_JSON(clc))
        VERIFY(clc.beneficiary_aid == aid.aid_id, "CLC must be issued to requesting AID")
        VERIFY(clc.validity.not_before <= NOW(), "CLC not yet valid")
        VERIFY(clc.validity.not_after > NOW(), "CLC already expired")
        credential_plaintext = ECDH_ES_DECRYPT(clc.credential_envelope, private_key)
        STORE_CLC(clc)
        STORE_CREDENTIAL_IN_HSM(clc.clc_id, credential_plaintext)
        SCHEDULE_ROTATION(clc.clc_id, clc.rotation_policy.rotation_interval_seconds)
        SCHEDULE_IDLE_TIMEOUT(clc.clc_id, clc.validity.idle_timeout_seconds)
        LOG_AUDIT_EVENT("clc_received", clc.clc_id, aid.aid_id)
        RETURN credential_plaintext

    ELIF response.body.status == "pending_human_approval":
        approval_token = response.body.approval_token
        RETURN A.POLL_APPROVAL(approval_token, timeout=response.body.estimated_review_seconds * 2)

    ELIF response.body.status == "rejected":
        LOG_REJECTION(response.body.error)
        RAISE CredentialRequestRejected(response.body.error.message)

function A.POLL_APPROVAL(approval_token, timeout):
    deadline = NOW() + timeout
    WHILE NOW() < deadline:
        SLEEP(30)  -- MUST poll no more frequently than every 30 seconds
        response = HTTP_GET(HUB_URL + "/agex/v1/credentials/approval/" + approval_token)
```

```
            IF response.body.status == "approved":
                RETURN A.HANDLE_CREDENTIAL_RESPONSE({status: 200, body: response.body})
            ELIF response.body.status IN ["rejected", "timed_out"]:
                RAISE ApprovalFailed(response.body.status)
        RAISE ApprovalTimeout()

    -- HUB PROCESSING --
    function H.PROCESS_CREDENTIAL_REQUEST(aid, manifest, request_headers):
        VERIFY_TIMESTAMP(request_headers.timestamp, tolerance=300)
        VERIFY_NO_DUPLICATE_REQUEST_ID(request_headers.request_id, window=86400)
        VERIFY_AID_NOT_EXPIRED(aid)
        VERIFY_AID_NOT_REVOKED(aid.aid_id)
        VERIFY_ED25519(aid.ia_signature, IA_PUBLIC_KEY[aid.issuer.ia_id], CANONICAL_JSON(aid))
        VERIFY_ED25519(manifest.agent_signature, aid.public_key, CANONICAL_JSON(manifest))
        VERIFY(manifest.requesting_aid == aid.aid_id, "Manifest AID must match request AID")
        sp = LOOKUP_SERVICE_PROVIDER(manifest.target.service_id)
        IF sp == null: RETURN ERROR("SERVICE_NOT_FOUND")
        FOR scope IN manifest.target.requested_scopes:
            IF scope NOT IN sp.available_scopes: RETURN ERROR("SCOPE_NOT_AVAILABLE")
        policy    = LOAD_APL_POLICY(sp.service_id)
        decision  = H.EVALUATE_APL(policy, aid, manifest)
        IF decision.action == "approve":
            RETURN H.FORWARD_TO_SP(sp, aid, manifest)
        ELIF decision.action == "review":
            approval_token = H.CREATE_PENDING_REVIEW(sp, aid, manifest, decision.params)
            NOTIFY_HUMAN_APPROVER(decision.params.notify, approval_token)
            RETURN {status: "pending_human_approval", approval_token: approval_token,
                    estimated_review_seconds: decision.params.review_timeout_seconds}
        ELIF decision.action == "reject":
            LOG_REJECTION(aid.aid_id, manifest.manifest_id, decision.params.reason)
            RETURN ERROR("REQUEST_REJECTED", decision.params.reason)
```

8.2 Autonomous Rotation (ARP)

```
-- AUTONOMOUS ROTATION PROTOCOL (ARP) --

function A.EXECUTE_ROTATION(clc_id):
    clc         = LOAD_CLC(clc_id)
    private_key = LOAD_PRIVATE_KEY_FROM_HSM()
    VERIFY(clc.validity.not_after > NOW() + 120,
           "CLC must not expire before rotation can complete")
    VERIFY(NOT CLC_IS_REVOKED(clc_id))

    new_keypair = ED25519_GENERATE_KEYPAIR()  -- MUST use fresh entropy

    rotation_request = {
        arp_version:       "1.0",
        message_type:      "rotation_request",
        request_id:        UUID4(),
        clc_id:            clc_id,
        requesting_aid:    clc.beneficiary_aid,
        rotation_reason:   "scheduled",
        new_public_key:    new_keypair.public,
        requested_validity_seconds: clc.rotation_policy.rotation_interval_seconds,
        nonce:             RANDOM_BYTES(16),
        timestamp:         NOW()
    }
    rotation_request.agent_signature = ED25519_SIGN(
        CANONICAL_JSON(rotation_request), private_key)

    response = HTTP_POST(HUB_URL + "/agex/v1/credentials/rotate", rotation_request)

    IF response.status == 200:
        new_clc    = response.body.new_clc
        old_expiry = response.body.old_clc_expiry
        VERIFY_ED25519(new_clc.provider_signature, SP_PUBLIC_KEY, CANONICAL_JSON(new_clc))
        VERIFY_ED25519(new_clc.hub_binding.hub_signature, HUB_PUBLIC_KEY, CANONICAL_JSON(new_clc))
        new_cred = ECDH_ES_DECRYPT(new_clc.credential_envelope, new_keypair.private)

        -- ATOMIC SWAP: two-phase commit
        -- Phase 1: Stage new credential
        STORE_NEW_CLC(new_clc)
        STORE_CREDENTIAL_IN_HSM(new_clc.clc_id, new_cred)

        -- Phase 2: Limit old credential validity
        -- MUST be atomic: if phase 2 fails, roll back phase 1
```

```
            TRY:
                SET_CLC_EXPIRY(clc_id, old_expiry)
                UPDATE_AID_KEYPAIR(new_keypair)  -- Switch to new key
            CATCH:
                -- Rollback: delete staged new credential
                DELETE_CLC(new_clc.clc_id)
                DELETE_CREDENTIAL_FROM_HSM(new_clc.clc_id)
                RAISE RotationAtomicSwapFailed()

            SCHEDULE_ROTATION(new_clc.clc_id,
                new_clc.rotation_policy.rotation_interval_seconds)
            LOG_AUDIT_EVENT("rotation_completed", clc_id, new_clc.clc_id)
            RETURN new_cred

        ELIF response.status == 409:
            WAIT_AND_RETRY(delay=EXPONENTIAL_BACKOFF())

        ELSE:
            LOG_ROTATION_FAILURE(clc_id, response.body.error)
            IF TIME_TO_EXPIRY(clc_id) < 0.1 * clc.rotation_policy.rotation_interval_seconds:
                ALERT_HUMAN_PRINCIPAL("CLC approaching expiry with rotation failures")
            SCHEDULE_RETRY(delay=EXPONENTIAL_BACKOFF())
```

8.3 Scope Escalation

```
-- SCOPE ESCALATION FLOW --

function A.REQUEST_ESCALATION(clc_id, additional_scopes, justification, task_id):
    clc         = LOAD_CLC(clc_id)
    private_key = LOAD_PRIVATE_KEY_FROM_HSM()
    VERIFY(NOT CLC_IS_REVOKED(clc_id))

    -- All additional scopes must be within scope ceiling
    FOR scope IN additional_scopes:
        IF scope NOT IN clc.scope_ceiling:
            RAISE ScopeCeilingExceeded(scope)

    union_scopes = UNION(clc.granted_scopes, additional_scopes)
    escalation_request = {
        message_type:      "escalation_request",
        request_id:        UUID4(),
        clc_id:            clc_id,
        requesting_aid:    clc.beneficiary_aid,
        additional_scopes: additional_scopes,
        target_scopes:     union_scopes,
        justification:     justification,
        task_id:           task_id,
        timestamp:         NOW()
    }
    escalation_request.agent_signature = ED25519_SIGN(
        CANONICAL_JSON(escalation_request), private_key)

    response = HTTP_POST(HUB_URL + "/agex/v1/credentials/escalate", escalation_request)

    IF response.body.status == "approved":
        updated_clc = response.body.updated_clc
        VERIFY_SIGNATURES(updated_clc)
        VERIFY(SUBSET(additional_scopes, updated_clc.granted_scopes),
               "Approved CLC must contain requested scopes")
        STORE_CLC(updated_clc)  -- Replaces old CLC
        LOG_AUDIT_EVENT("escalation_approved", clc_id, additional_scopes)
        RETURN updated_clc

-- HUB ESCALATION PROCESSING --
function H.EVALUATE_ESCALATION(request):
    clc = LOAD_CLC(request.clc_id)
    aid = LOAD_AID(request.requesting_aid)

    FOR scope IN request.additional_scopes:
        IF scope NOT IN clc.scope_ceiling:
            RETURN ERROR("SCOPE_CEILING_EXCEEDED")
    IF aid.trust_tier >= 2:
        RETURN H.FORWARD_ESCALATION_TO_SP(request, clc)
    ELSE:
        RETURN H.ROUTE_ESCALATION_TO_HUMAN(request, clc)
```

8.4 Delegation Chain Creation

```
-- DELEGATION CHAIN CREATION FLOW --

function A.CREATE_DELEGATION(parent_clc_id, child_aid_id, sub_scopes,
                              duration_seconds, purpose, max_further_depth):
    parent_clc  = LOAD_CLC(parent_clc_id)
    private_key = LOAD_PRIVATE_KEY_FROM_HSM()
    VERIFY(NOT CLC_IS_REVOKED(parent_clc_id))
    VERIFY(parent_clc.delegation_policy.delegation_permitted)
    FOR scope IN sub_scopes:
        IF scope NOT IN parent_clc.granted_scopes:
            RAISE ScopeCeilingExceeded(scope)
    VERIFY(duration_seconds <= TIME_TO_EXPIRY(parent_clc_id))

    current_depth = LEN(parent_clc.chain_provenance) + 1
    VERIFY(current_depth <= parent_clc.delegation_policy.max_delegation_depth)

    delegation_request = {
        delegation_version:   "1.0",
        message_type:         "delegation_request",
        delegation_id:        UUID4(),
        parent_clc_id:        parent_clc_id,
        parent_aid:           parent_clc.beneficiary_aid,
        child_aid:            child_aid_id,
        requested_sub_scopes: sub_scopes,
        delegation_duration_seconds: duration_seconds,
        chain_depth:          current_depth,
        max_further_delegation_depth: max_further_depth,
        purpose:              purpose,
        timestamp:            NOW()
    }
    delegation_request.parent_signature = ED25519_SIGN(
        CANONICAL_JSON(delegation_request), private_key)

    response = HTTP_POST(HUB_URL + "/agex/v1/credentials/delegate", delegation_request)
    IF response.body.status == "delegated":
        sub_clc = response.body.sub_clc
        VERIFY_SIGNATURES(sub_clc)
        VERIFY(SUBSET(sub_scopes, sub_clc.granted_scopes))
        VERIFY(sub_clc.chain_provenance[-1] == parent_clc_id)
        LOG_AUDIT_EVENT("delegation_created", parent_clc_id, sub_clc.clc_id)
        RETURN sub_clc
```

```
-- HUB DELEGATION VALIDATION --
function H.VALIDATE_DELEGATION(request):
    parent_clc = LOAD_CLC(request.parent_clc_id)
    VERIFY_ACTIVE(parent_clc)
    VERIFY_ED25519(request.parent_signature, LOAD_AID(request.parent_aid).public_key,
                   CANONICAL_JSON(request))
    FOR scope IN request.requested_sub_scopes:
        IF scope NOT IN parent_clc.granted_scopes:
            RETURN ERROR("SCOPE_CEILING_EXCEEDED", scope)
    child_aid = LOAD_AID(request.child_aid)
    VERIFY_AID_NOT_EXPIRED(child_aid)
    VERIFY_AID_NOT_REVOKED(child_aid.aid_id)
    VERIFY(request.chain_depth <= parent_clc.delegation_policy.max_delegation_depth)
    VERIFY(H.COUNT_ACTIVE_SUB_CLCS(request.parent_clc_id) <
           parent_clc.delegation_policy.max_concurrent_sub_clcs)
    -- Issue sub-CLC with scope ceiling == granted scopes (prevents further escalation)
    sub_clc = H.ISSUE_SUB_CLC(parent_clc, child_aid, request)
    sub_clc.chain_provenance = parent_clc.chain_provenance + [request.parent_clc_id]
    sub_clc.scope_ceiling    = request.requested_sub_scopes
    sub_clc.granted_scopes   = request.requested_sub_scopes
    RETURN sub_clc
```

8.5 Emergency Revocation System

```
-- EMERGENCY REVOCATION SYSTEM (ERS) --

function INITIATOR.INITIATE_ERS(target_aid_id, reason, initiator_credentials):
    VERIFY_AUTHORISED_TO_REVOKE(initiator_credentials, target_aid_id)
    ers_signal = {
        message_type:   "ers_signal",
        signal_id:      UUID4(),
        target_aid:     target_aid_id,
        reason:         reason,
        initiated_by:   initiator_credentials.identity,
        initiator_type: initiator_credentials.type,
        cascade:        true,
        timestamp:      NOW()
    }
    ers_signal.initiator_signature = ED25519_SIGN(
        CANONICAL_JSON(ers_signal), initiator_credentials.private_key)
    response = HTTP_POST(HUB_URL + "/agex/v1/ers/signal", ers_signal)
    RETURN response

-- HUB ERS PROCESSING --
function H.PROCESS_ERS(ers_signal):
    start_time = NOW()
    VERIFY_SIGNATURE(ers_signal)
    VERIFY_INITIATOR_AUTHORISED(ers_signal.initiated_by,
                                 ers_signal.initiator_type,
                                 ers_signal.target_aid)

    -- Find all active CLCs for target AID
    direct_clcs = DB.QUERY(
        "SELECT * FROM clcs WHERE beneficiary_aid = ? AND status = 'active'",
        ers_signal.target_aid)

    -- Find all descendant CLCs in delegation chains
    all_clcs = direct_clcs.copy()
    queue    = direct_clcs.copy()
    WHILE queue NOT EMPTY:
        parent_clc = queue.pop()
        children   = DB.QUERY(
            "SELECT * FROM clcs WHERE chain_provenance @> ARRAY[?] AND status = 'active'",
            parent_clc.clc_id)
        all_clcs.extend(children)
        queue.extend(children)
  -- Notify all service providers in parallel (MUST complete within 60 seconds)
  notifications = []
  FOR clc IN all_clcs:
      sp = LOOKUP_SERVICE_PROVIDER(clc.issuer_service_id)
      notifications.append(
          ASYNC_HTTP_POST(sp.agex_endpoint + "/revoke",
              {clc_id: clc.clc_id, signal_id: ers_signal.signal_id,
               revoke_at: NOW()}))

  AWAIT_ALL(notifications, timeout=50)  -- 50s to allow 10s margin

  -- Bulk revoke in Hub database
  DB.BULK_UPDATE(all_clcs,
      {status: "revoked",
       revoked_at: NOW(),
       revocation_reason: ers_signal.reason,
       ers_signal_id: ers_signal.signal_id})

  elapsed = NOW() - start_time
  ASSERT(elapsed < 60, "ERS completion bound exceeded: " + elapsed + "s")

  LOG_ERS_AUDIT_EVENT(ers_signal, all_clcs, elapsed)
  RETURN {
      status: "revocation_initiated",
      clcs_revoked: LEN(direct_clcs),
      sub_clcs_revoked: LEN(all_clcs) - LEN(direct_clcs),
      services_notified: LEN(SET(clc.issuer_service_id FOR clc IN all_clcs)),
      completion_time_ms: elapsed * 1000
  }
```

9. Formal Security Analysis

9.1 Credential Confidentiality (SR-1)

Theorem: Theorem: Under the IND-CCA2 security of AES-256-GCM and the CDH hardness assumption for Curve25519, AGEX satisfies SR-1.

Proof sketch: The credential_envelope uses ECDH-ES key agreement between the service provider and the agent's Ed25519 public key (converted to X25519 per RFC 8037). The shared secret is used via HKDF-SHA256 to derive a 256-bit AES-GCM key. For any party other than the beneficiary agent to decrypt the envelope, they must either (a) possess the agent's Ed25519 private key, (b) solve CDH on Curve25519 (assumed hard), or (c) break AES-256-GCM IND-CCA2 (assumed hard with 256-bit keys). The Hub receives the CLC but never the agent's private key. Therefore SR-8 (Hub Zero-Knowledge) follows as a corollary.

9.2 Forward Secrecy Under Rotation (SR-3)

Theorem: Theorem: AGEX satisfies SR-3 if agents generate fresh Ed25519 keypairs each rotation cycle and the atomic swap is correctly implemented.

Proof sketch: ARP requires a fresh keypair (new_keypair) before rotation. The new CLC is encrypted under new_keypair.public. After the atomic swap, the old private key is deleted. An adversary obtaining new_keypair.private learns only the new credential. Old credentials were encrypted under the old public key; recovery requires CDH on Curve25519 from the old public key alone, assumed infeasible. The atomic swap's two-phase commit ensures no window in which neither credential is valid.

9.3 Delegation Scope Monotonicity (SR-4)

Theorem: Theorem: AGEX satisfies SR-4 if the Hub correctly validates delegation requests and the scope ceiling is enforced.

Proof sketch: Hub validation rejects any delegation request with scopes not present in the parent CLC's granted_scopes. The issued sub-CLC sets scope_ceiling equal to granted_scopes, preventing escalation. Expanding scope through delegation requires either compromising the Hub, forging the service provider's Ed25519 signature on a elevated-scope CLC, or forging the Hub's signature on such a CLC. All three require breaking Ed25519 EUF-CMA, assumed infeasible.

9.4 Cascade Revocation Completeness (SR-5)

Theorem: Theorem: AGEX satisfies SR-5 if the Hub correctly maintains the chain_provenance index and the 60-second bound is met.

Proof sketch: Every sub-CLC carries chain_provenance listing all ancestor CLC IDs. The Hub maintains an index from CLC ID to all descendants. ERS queries this index to find all chain descendants. By induction: at depth 0, the target AID's CLCs are revoked. At depth d+1, CLCs whose chain_provenance contains a depth-d CLC are revoked. The maximum depth is bounded by max_delegation_depth. The 60-second bound requires parallel SP notification and is an operational requirement enforced by the HC-M-008 conformance requirement.

9.5 Attack Analysis

Theorem: Seven principal attack vectors are analysed below.

Proof sketch: Attack 1 (Credential Interception): All Hub communications use TLS 1.3. Additionally, credential envelopes are independently encrypted under agent-specific keys, providing defence in depth. Attack 2 (AID Forgery): AIDs are signed by certified IAs. Forging a valid AID signature requires compromising an IA private key or breaking Ed25519. Attack 3 (Manifest Tampering): Manifests carry agent_signature; any modification invalidates the signature, which the SP verifies. Attack 4 (Request Replay): Each request carries a unique request_id enforced by 24-hour deduplication, and timestamp validation within 300 seconds. Attack 5 (Scope Escalation via Delegation): Hub validates all delegation requests against parent CLC granted_scopes before issuing sub-CLCs. Attack 6 (ERS Evasion): CLCs issued after ERS signal timestamp for a revoked AID are rejected; SPs check revocation on every API call. Attack 7 (Hub Insider): Zero-knowledge envelope design means Hub database contains only ciphertext; insider with full DB access cannot decrypt credentials.

10. Comparative Evaluation

We evaluate AGEX against three existing protocols across twelve criteria relevant to autonomous agent credential management. Ratings: Full (2 points), Partial (1 point), None (0 points).

Criterion	OAuth 2.0	OIDC	SPIFFE	AGEX
C1: Agent-native identity	None	None	Partial	Full
C2: Autonomous credential acquisition	Partial	Partial	Full	Full
C3: Credential lifecycle management	Partial	Partial	Partial	Full
C4: Autonomous rotation	None	None	Full	Full
C5: Dynamic scope escalation	None	None	None	Full
C6: Delegation with scope reduction	None	None	None	Full
C7: Cross-organisational federation	Full	Full	Partial	Full
C8: Structured intent declaration	None	None	None	Full
C9: Emergency cascade revocation	None	None	None	Full
C10: Zero-knowledge credential brokering	N/A	N/A	N/A	Full
C11: Standardised audit trail	None	None	Partial	Full
C12: Human override guarantee	None	None	None	Full
Total Score (max 24)	5	5	8	24

OAuth 2.0 and OIDC score 5/24, achieving only partial marks on credential acquisition, lifecycle management, and federation. SPIFFE scores 8/24, performing well on autonomous rotation and acquisition but providing no delegation, intent declaration, or cross-service revocation. AGEX achieves 24/24, the only protocol addressing all twelve criteria. This evaluation confirms that AGEX is not an incremental improvement but a purpose-built solution to requirements no existing protocol addresses.

11. Worked Deployment Scenarios

Six worked deployment scenarios demonstrate AGEX applied to representative real-world architectures. Each scenario illustrates different aspects of the protocol and highlights the operational benefits of the AGEX approach relative to current credential management practices.

Scenario 1: Single-Agent SaaS Integration

Organisation Acme Ltd deploys a single AI agent, AnalystBot, to pull data from a third-party analytics SaaS platform, generate weekly reports, and write summaries to Acme's internal document store. This is the simplest AGEX deployment pattern and serves as the baseline for understanding the protocol in practice.

AnalystBot is registered as a Tier 1 agent (verified organisational identity) with the AGEX Hub operated by the analytics SaaS provider. Its AID declares type: worker, capabilities: [data_extraction, report_generation], and a principal identifying Acme Ltd.

The weekly report generation task initiates as follows. AnalystBot constructs an Intent Manifest declaring intent.task_type: read, intent.data_classification: internal, intent.automated: true, duration.max_duration_seconds: 7200 (2 hours), and target.requested_scopes: [analytics:read, reports:export]. The manifest's escalation thresholds are set to human review if data_records_accessed exceeds 10,000, reflecting Acme's data governance policy.

The SaaS provider's APL policy auto-approves Tier 1 agents requesting analytics:read and reports:export with data_classification: internal and max_duration_seconds under 86400. AnalystBot's request is auto-approved in approximately 200ms round-trip.

The issued CLC carries a rotation_interval_seconds of 3600, meaning ARP fires approximately 55 minutes into the 2-hour task. AnalystBot's rotation scheduler detects the upcoming rotation interval, generates a new Ed25519 keypair, and submits a rotation request to the Hub with rotation_reason: scheduled. The Hub coordinates with the SaaS provider, issues a new CLC, and AnalystBot performs the atomic swap. The task continues without interruption; the SaaS API sees authenticated requests throughout with no gap.

Upon task completion, AnalystBot does not hold an open CLC: the CLC's idle_timeout_seconds fires 300 seconds after the last API call, triggering automatic revocation. The Audit Ledger records the full lifecycle: issuance at T+0, rotation at T+3600, idle timeout revocation at T+7500. Acme's security team can query this record at any time via the audit API.

Scenario 2: Multi-Agent Orchestration with Delegation

Organisation Nexus Corp runs a customer onboarding automation system using a three-tier agent architecture. The Orchestrator agent plans and coordinates the overall workflow. Four Worker agents handle specific sub-tasks: KYC verification, account creation, welcome email dispatch, and CRM record creation. Each Worker agent needs different API credentials for different services.

The Orchestrator is registered as a Tier 2 agent with broad but bounded permissions. It holds a CLC for the internal orchestration service with scopes including delegate:kyc, delegate:accounts, delegate:email, delegate:crm. These scopes do not grant the Orchestrator direct access to the underlying services; they grant it the right to delegate access to its Worker agents.

When a new customer onboarding task begins, the Orchestrator submits four delegation requests in parallel, one per Worker agent. Each request specifies: - parent_clc_id: the Orchestrator's CLC - child_aid: the relevant Worker's AID - requested_sub_scopes: only the scopes relevant to that Worker's task - delegation_duration_seconds: 1800 (30 minutes, the expected task window) - max_further_delegation_depth: 0 (Workers may not further delegate)

The Hub validates each request: confirms the Orchestrator's CLC is valid, confirms each requested scope is a subset of the Orchestrator's granted scopes, confirms each Worker's AID is valid and non-revoked, and issues four sub-CLCs, each with a chain_provenance containing the Orchestrator's CLC ID.

If the KYC service flags the customer as high-risk mid-workflow, the Orchestrator calls the ERS endpoint targeting its own CLC. The Hub cascades revocation to all four sub-CLCs within 15 seconds. All Worker agents lose their credentials simultaneously. The Orchestrator logs the revocation event with the task_id, creating a complete audit trail for the compliance team.

This scenario demonstrates that AGEX's delegation model provides fine-grained access control in complex agent systems without requiring each Worker agent to independently negotiate credentials with each service.

Scenario 3: Cross-Organisational Agent Federation

TravelCo operates an AI travel booking agent that its customers use via a mobile application. The booking agent (TravelBot) needs to call credential-protected APIs operated by three independent organisations: HotelChain, AirlineX, and PaymentCo. Each of these organisations operates its own AGEX Hub or has registered services with a shared regional Hub.

TravelBot's AID is issued by TravelCo's corporate Identity Authority and registered with TravelCo's Hub (Hub-T). HotelChain and AirlineX have services registered with the travel industry's shared Hub (Hub-I). PaymentCo operates its own Hub (Hub-P).

When a customer initiates a booking, TravelBot submits credential requests to Hub-T for all three services. Hub-T resolves each request: for HotelChain and AirlineX, Hub-T identifies that these services are registered with Hub-I and initiates a cross-Hub attestation. For PaymentCo, Hub-T initiates a separate cross-Hub attestation to Hub-P.

Both Hub-I and Hub-P receive Cross-Hub Attestations from Hub-T, verify Hub-T's certificate against the AGEX Root of Trust, and evaluate their respective APL policies. Hub-I's policy auto-approves TravelBot (a Tier 2 agent from a known travel operator) for hotel and flight search scopes. Hub-P's policy routes the payment write scope to human review given TravelBot's Tier 2 status and the payment risk profile.

TravelBot receives CLCs for hotel and flight search immediately and begins searching. It polls the approval status endpoint for the payment CLC. PaymentCo's payment operations manager reviews and approves the request within 4 minutes. TravelBot proceeds to hold the customer's payment method.

From the customer's perspective, the booking takes approximately 5 minutes, most of which is human approval time for the payment CLC. The hotel and flight search run in parallel with the approval wait. From TravelBot's perspective, it holds three CLCs from three different originating services, all brokered through Hub-T, with a single consistent audit trail per CLC covering the full lifecycle.

Scenario 4: Emergency Incident Response

SecurityCo operates an AI-powered security operations platform. Its incident response agent, IRAgent, holds CLCs for 23 different services including logging platforms, network management APIs, cloud provider control planes, and communication services. IRAgent is a Tier 3 certified agent with broad autonomous authority within defined escalation thresholds.

At 03:47 UTC, IRAgent detects anomalous behaviour consistent with a credential compromise: it observes API calls being made to services it holds credentials for from IP addresses it did not initiate, at rates exceeding its own anomaly thresholds. IRAgent cannot determine whether its own credentials have been compromised or whether the target services are being attacked via a separate path.

IRAgent initiates self-revocation via the ERS endpoint with reason: compromise_suspected, targeting its own AID. Within 8 seconds, the Hub cascades revocation to all 23 CLCs. The Audit Ledger records the self-initiated ERS event with the anomaly data IRAgent attached as context.

Simultaneously, the ERS signal is forwarded to SecurityCo's security operations centre. Human analysts review the audit data and determine that two of the 23 CLCs had been used from unauthorised addresses; the anomaly detection was correct. They complete a root cause analysis, re-issue a clean AID for IRAgent under a new keypair, and selectively re-request CLCs for the 21 services that were unaffected.

For the two compromised services, SecurityCo initiates their own incident response procedures. The AGEX audit record provides a complete, cryptographically verifiable account of which credentials were used, when, from which addresses (via service provider logs cross-referenced against the CLC IDs), and when they were revoked.

This scenario demonstrates that AGEX's ERS mechanism enables a compromised agent to contain its own damage, and that the structured audit trail supports post-incident forensic analysis.

Scenario 5: Regulatory Compliance Workflow

FinanceCo operates AI agents that process customer financial data under GDPR and PCI DSS obligations. Their compliance team uses AGEX's data handling fields and APL policies to enforce regulatory requirements at the credential layer.

For any credential request involving PII processing (manifest.data_handling.pii_processing: true), FinanceCo's APL policy requires: data_handling.retention_policy is no_retention or session (not persistent), data_handling.deletion_on_completion is true, data_handling.cross_border_transfer is false unless transfer_jurisdictions contains only approved jurisdictions, and trust_tier is at least 2.

For PCI DSS scope, any credential request including payment data scopes requires: trust_tier is 3, intent.data_classification is restricted, and the requesting agent's AID has a PCI-DSS capability declaration verified by FinanceCo's IA.

These requirements are encoded as APL rules and evaluated automatically by the Hub Policy Engine for every credential request. Agents that do not meet the requirements receive a rejection with a structured error response identifying which policy rule was violated. The APL policy document itself is version-controlled and its hash is recorded in the Audit Ledger, providing a tamper-evident record of which policy version governed each CLC issuance.

FinanceCo's compliance team can produce an audit report for any time period showing: every CLC issued for PII-touching operations, the exact policy version in force at each issuance, the intent declared by the requesting agent, the scopes granted, the data handling commitments accepted, and the revocation events. This report is produced by querying the AGEX audit API with a date range filter and is available in structured JSON for automated compliance tooling.

Scenario 6: Consumer-Facing Agent Service

AssistantCo operates a consumer AI assistant that acts on behalf of individual users across dozens of third-party services: email, calendar, banking, shopping, social media, and productivity tools. Each user's assistant instance is a distinct agent with a distinct AID, registered under AssistantCo's organisation but with the individual user identified in the AID's principal.jurisdiction and principal.contact fields.

Scale is the defining characteristic of this deployment: AssistantCo manages tens of thousands of distinct agent instances, each potentially holding multiple CLCs. The operational automation of credential management is not optional; it is the only viable approach at this scale.

When a user grants their assistant access to a new service, the assistant submits a credential request. Because the user is a consumer (not an enterprise with a pre-established trust relationship), the agent begins at Tier 0 with every new service. The manifest includes intent fields that are visible to the service provider and, via the service provider's user dashboard, to the end user. The user can see exactly what their assistant declared it would do with access to the service.

The service provider's APL policy for consumer agents requires explicit user confirmation for write scopes. The Hub returns a pending_human_approval response with an approval URL that AssistantCo renders in the user interface. The user approves (or denies) via the interface. This is the one moment of synchronous human involvement in an otherwise automated lifecycle.

Once approved, subsequent rotations, scope escalations within pre-approved bounds, and eventual revocation when the user withdraws consent are all handled automatically by the AGEX infrastructure. When the user asks their assistant to stop accessing a service, the assistant calls the ERS endpoint. The service provider revokes the CLC within seconds, and the user's consent withdrawal is complete.

This scenario demonstrates that AGEX's architecture accommodates both enterprise batch deployments and consumer-scale individual agent instances, with a consistent protocol but appropriately different operational patterns.

12. Conformance Testing Requirements

A conformance testing framework defines the minimum requirements for an AGEX implementation to be considered conformant with this specification. Conformance is assessed separately for three implementation roles: Agent Implementation, Hub Implementation, and Service Provider Implementation. An implementation may conform in one, two, or all three roles.

Conformance is binary: an implementation either passes all mandatory requirements for a role or it does not. There are no partial conformance grades. Implementations that pass all mandatory requirements for a role and additionally pass optional requirements SHOULD advertise which optional requirements they satisfy.

The conformance test suite is published and maintained by the AGEX Foundation as an open source test harness at tests.agex.api. The test harness provides both automated tests (for protocol message validation, cryptographic correctness, and API contract adherence) and documented manual test procedures (for operational requirements such as the 60-second ERS completion bound and human escalation routing).

12.1 Agent Implementation Requirements

Agent Implementation Conformance Requirements

Mandatory (MUST)

```
AC-M-002: Compute AID request signatures using Ed25519 over RFC 8785 canonical JSON.
MUST verify the correctness of signatures before submission.

AC-M-003: Implement Intent Manifest construction conforming to the full schema defined
in Section 4.2, including all REQUIRED fields. MUST compute and include
agent_signature over the canonical manifest.

AC-M-004: Decrypt CLC credential envelopes using ECDH-ES with the agent's private key.
MUST verify provider_signature and hub_signature on received CLCs before
decrypting or using credential material.

AC-M-005: Implement ARP rotation scheduling. MUST initiate rotation before
rotation_interval_seconds expires, with sufficient lead time to complete
rotation before CLC expiry. Recommended lead time: 15% of rotation interval.

AC-M-006: Implement the ARP atomic swap as a two-phase operation. MUST NOT discard
```

```
old credential material until the new CLC has been successfully decrypted
and stored.
```

```
AC-M-007: Generate fresh Ed25519 keypairs for each rotation cycle. MUST NOT reuse
private key material across rotation cycles.
```

```
AC-M-008: Check CLC validity (not_before, not_after, revocation status) before
using credential material. MUST NOT use credentials from a revoked or
expired CLC.
```

```
AC-M-009: Implement exponential backoff for failed rotation attempts. MUST alert
the human principal if rotation fails and the CLC is within 10% of its
validity period.
```

```
AC-M-010: Include X-AGEX-Request-ID, X-AGEX-Timestamp, X-AGEX-AID, and
X-AGEX-Signature in all Hub API requests per Section 5.
```

Optional (SHOULD)

```
AC-O-002: Implement self-initiated ERS for anomaly detection scenarios.
```

```
AC-O-003: Implement structured logging of all credential lifecycle events to the
audit endpoint specified in the CLC.
```

```
AC-O-004: Implement idle timeout detection and voluntary CLC release when
idle_timeout_seconds is specified in the CLC.
```

12.2 Hub Implementation Requirements

Hub Implementation Conformance Requirements

Mandatory (MUST)

```
HC-M-002: Check AID validity (expiry, revocation) before processing any request
referencing an AID. MUST maintain a revocation index updated within
60 seconds of receiving a Revocation Record.
```

```
HC-M-003: Verify manifest agent_signature before forwarding to service providers.
```

```
HC-M-004: Implement APL policy evaluation for all credential requests. MUST correctly
evaluate all APL condition types defined in Section 6.
```

```
HC-M-005: Implement credential envelope zero-knowledge architecture. MUST NOT attempt
to decrypt credential envelopes. MUST NOT store credential plaintext.
```

```
HC-M-006: Implement the ARP flow as defined in Section 5.2, including the atomic swap
guarantee and overlap period.
```

```
HC-M-007: Implement delegation request validation per Section 4.5. MUST reject
delegation requests where requested_sub_scopes are not a strict subset of
the parent CLC's granted_scopes.
```

```
HC-M-008: Implement the ERS endpoint. MUST complete cascade revocation within 60
seconds of receiving a valid ERS signal. MUST record the completion time
in the Audit Ledger.
```

```
HC-M-009: Implement a tamper-evident Audit Ledger. MUST hash each event with SHA-3-256
and chain events such that modification is detectable. MUST provide the
audit query API defined in Section 5.
```

HC-M-010: Enforce request ID deduplication within a 24-hour window. MUST reject duplicate request IDs.

HC-M-011: Enforce timestamp validation. MUST reject requests where X-AGEX-Timestamp is more than 300 seconds from Hub current time.

HC-M-012: Implement rate limiting. MUST return 429 Too Many Requests with a Retry-After header when rate limits are exceeded.

HC-M-013: Implement TLS 1.3 for all Hub API endpoints. MUST NOT accept TLS 1.2 or earlier connections.

Optional (SHOULD)

HC-O-002: Implement the service provider discovery endpoint.

HC-O-003: Implement webhook notifications to service providers and agents for lifecycle events (rotation, escalation, revocation).

HC-O-004: Implement anomaly detection on credential usage patterns and alert service providers when anomaly thresholds defined in CLCs are exceeded.

HC-O-005: Implement geographic routing for data residency compliance.

12.3 Service Provider Requirements

Service Provider Implementation Conformance Requirements

Mandatory (MUST)

SC-M-002: Issue CLCs conforming to the full schema defined in Section 4.3. MUST include hub_binding and provider_signature.

SC-M-003: Implement credential envelope encryption using ECDH-ES+AES256GCM with the beneficiary agent's public key. MUST NOT transmit credential plaintext outside the encrypted envelope.

SC-M-004: Honour ARP rotation requests from the Hub. MUST issue replacement CLCs within 30 seconds of receiving a valid rotation_forwarding message.

SC-M-005: Implement CLC revocation on receiving a revocation signal from the Hub. MUST invalidate the credential identified by the CLC ID within 5 seconds of receiving the signal.

SC-M-006: Honour the ERS signal from the Hub. MUST revoke all credentials associated with the target AID within 10 seconds of receiving the signal.

SC-M-007: Verify CLC validity (not_before, not_after) and revocation status on every authenticated API call. MUST NOT honour credentials from expired or revoked CLCs.

SC-M-008: Publish an APL policy document at the policy_endpoint registered with the Hub. MUST keep the policy document current and notify the Hub of updates.

Optional (SHOULD)

SC-O-002: Publish a service metadata document in the format defined by the Hub discovery endpoint specification.

SC-O-003: Implement usage reporting to the Hub at the interval defined in the CLC's audit_requirements.

13. Privacy and Compliance Analysis

AGEX processes data about agents and their activities that may, in some deployment contexts, relate to identifiable individuals (particularly in consumer-facing deployments described in Scenario 6) or to commercially sensitive business processes. This section analyses AGEX's privacy implications under the General Data Protection Regulation (GDPR), the California Consumer Privacy Act (CCPA) [CCPA], and relevant AI governance frameworks, and provides implementation guidance for compliant deployments.

This analysis is provided for informational purposes. Organisations deploying AGEX SHOULD obtain independent legal advice on the application of privacy regulations to their specific deployment context. The AGEX Foundation does not provide legal advice.

13.1 GDPR Analysis

GDPR Analysis

The primary GDPR question for AGEX deployments is whether AID data constitutes personal data under Article 4(1) GDPR. An AID contains the owning organisation's name and contact email, the agent's name (if provided), and the AGEX-assigned agent identifier. In enterprise deployments where agents are deployed by legal entities rather than individuals, AID data is unlikely to constitute personal data under GDPR (it relates to a company, not a natural person). In consumer deployments where each end user has a distinct agent instance (Scenario 6), the agent's AID is linked to the user's account and therefore constitutes personal data.

For consumer deployments, the following GDPR obligations apply:

Lawful basis: Processing AID data requires a lawful basis. The most applicable basis is Article 6(1)(b) (contract performance), where the agent service is the subject of a contract with the user, or Article 6(1)(a) (consent), where the user has consented to the agent's operation.

Data subject rights: Users have rights to access, rectification, erasure (the right to be forgotten), and portability of their AID data. ERS-initiated revocation does not constitute erasure of audit records; erasure of audit records is subject to the retention obligations discussed below.

Retention: AGEX recommends a 7-year retention period for credential lifecycle audit records. Under GDPR, retention periods must be justified by a legal basis. For regulated industries (financial services, healthcare), the 7-year period may be required by sector-specific regulation. For non-regulated deployments, shorter retention periods may be appropriate and SHOULD be specified in the Hub's privacy policy.

Data minimisation: The Intent Manifest's summary field SHOULD NOT contain personal data. The pii_categories field in data_handling is for classification purposes only and MUST NOT contain actual PII. Implementations MUST apply data minimisation to all fields that are not strictly necessary for the protocol's function.

Cross-border transfers: The manifest's data_handling.cross_border_transfer and transfer_jurisdictions fields enable organisations to declare their data transfer intentions at credential request time. Service providers whose APL policies prohibit cross-border transfers can enforce these prohibitions at the credential issuance layer. Hub operators offering AGEX services to EU-based users MUST ensure their Hub infrastructure is deployed within the EEA or that appropriate GDPR transfer mechanisms (Standard Contractual Clauses, adequacy decisions) are in place.

13.2 CCPA Analysis

CCPA Analysis

The California Consumer Privacy Act applies to businesses that meet specified thresholds and process personal information of California consumers. The CCPA's definition of personal information (Cal. Civ. Code Section 1798.140(o) [CCPA]) is broad and includes inferences drawn from data to create consumer profiles.

For AGEX deployments serving California consumers, the principal CCPA obligations are:

Right to know: Consumers have the right to know what personal information a business has collected and how it is used. AGEX's structured audit trail, accessible via the audit API, provides the technical foundation for responding to right-to-know requests. The audit API response format is machine-readable and can be used to generate consumer-facing reports.

Right to delete: The CCPA's right to delete covers personal information held by the business and by service providers acting on the business's behalf. ERS-initiated revocation of CLCs satisfies the operational component of deletion (the agent loses access), but audit records may need to be retained under exceptions for legal obligations. Implementations SHOULD clearly document which audit records are retained under legal obligation exceptions and which are deleted in response to consumer requests.

Service provider agreements: Under the CCPA, Hub operators processing personal information on behalf of an AssistantCo (in Scenario 6 terms) are service providers under the CCPA and require appropriate service provider agreements restricting use of personal information to specified purposes.

13.3 AI Governance Framework Alignment

AI Governance Framework Alignment

The NIST AI Risk Management Framework (AI RMF 1.0) [NIST-AI] defines four functions for managing AI risk: Govern, Map, Measure, and Manage. AGEX contributes to all four functions when applied to AI agent deployments.

In the Govern function, AGEX's APL policies provide a machine-enforceable layer of AI governance, translating organisational risk tolerance into automated credential approval rules. The policy version tracking in the Audit Ledger provides an auditable record of governance decisions over time.

In the Map function, the Intent Manifest's structured declaration of agent intent, data classification, reversibility, and human visibility provides a standardised vocabulary for mapping AI risks at the point of credential issuance. Service providers evaluating a manifest have explicit information about the risk profile of the requested operation before granting access.

In the Measure function, the Audit Ledger's structured event records provide quantitative data on agent credential usage patterns, escalation rates, anomaly detection events, and revocation frequencies. These metrics are directly relevant to AI system performance monitoring and risk measurement.

In the Manage function, the ERS provides a rapid containment mechanism for identified AI risks, enabling human principals to immediately revoke an agent's access across all services when a risk event is identified. The delegation chain mechanism enables fine-grained access management for multi-agent systems, reducing the blast radius of individual agent compromises.

The EU AI Act (Regulation 2024/1689) [EU-AIA], applicable from August 2026, imposes requirements on high-risk AI systems including those used in critical infrastructure, education, employment, and financial services. Article 9 of the EU AI Act [EU-AIA] requires a risk management system maintained throughout the

AI system lifecycle. AGEX's structured audit trail, version-controlled policies, and emergency revocation mechanisms are directly relevant to demonstrating compliance with this requirement. The AGEX Foundation intends to publish a dedicated EU AI Act compliance guidance document as the Act comes into force.

13.4 Data Residency Implementation Guidance

Data Residency Implementation Guidance

Many organisations operate under data residency requirements that restrict where certain data may be stored or processed. AGEX provides two mechanisms for addressing data residency at the protocol layer.

The AID's principal.jurisdiction field carries an ISO 3166-2 code identifying the jurisdiction of the owning organisation. This field enables Hub operators to route requests to regional Hub deployments based on the requesting organisation's jurisdiction.

The manifest's data_handling.transfer_jurisdictions field carries a list of ISO 3166-2 codes identifying the jurisdictions to which data may be transferred during the credentialed operation. Service providers whose APL policies include geographic conditions can automatically reject or escalate requests that would result in prohibited cross-border transfers.

Hub operators deploying AGEX infrastructure for organisations subject to EU data residency requirements SHOULD deploy Hub infrastructure with EEA-based data storage and processing. Hub operators deploying for organisations subject to Chinese data residency requirements (Cybersecurity Law, Data Security Law, PIPL) SHOULD deploy separate Hub infrastructure within China. The AHFP federation protocol enables cross-border Hub interoperability while maintaining data residency compliance at the infrastructure layer.

14. State Machine Specifications

This section formally specifies the state machines for all three AGEX protocol participants: the Agent, the Hub, and the Service Provider. State machine specifications are normative [RFC2119]; conformant implementations MUST implement the defined states and transitions. Undefined transitions (those not listed for a given state) MUST result in an error response or logging of an unexpected event, depending on context.

State machine diagrams are presented in textual notation. Each state is named and described. Transitions are denoted as: CURRENT_STATE --[EVENT / GUARD]--> NEXT_STATE with associated actions listed beneath. Guards are conditions that must be true for the transition to fire. Actions are operations executed atomically with the transition.

14.1 Agent State Machine

```
Agent State Machine

States: UNINITIALISED, AID_PENDING, AID_ACTIVE, REQUESTING_CREDENTIAL,
        APPROVAL_PENDING, CREDENTIAL_ACTIVE, ROTATING, ESCALATING,
        DELEGATING, SUSPENDED, REVOKED, DECOMMISSIONED

Initial state: UNINITIALISED

Transitions:

UNINITIALISED --[GENERATE_KEYPAIR / entropy_available]--> AID_PENDING
  Action: Generate Ed25519 keypair; store private key in HSM; submit AID request to IA.

AID_PENDING --[AID_ISSUED / ia_signature_valid]--> AID_ACTIVE
  Action: Store AID; register AID with Hub; schedule AID renewal alert at renewal_window_days.

AID_PENDING --[AID_REJECTED]--> UNINITIALISED
  Action: Log rejection reason; notify human principal.

AID_ACTIVE --[INITIATE_CREDENTIAL_REQUEST / service_id_known]--> REQUESTING_CREDENTIAL
  Action: Construct Intent Manifest; sign manifest; submit to Hub.

REQUESTING_CREDENTIAL --[CREDENTIAL_APPROVED / clc_signatures_valid]--> CREDENTIAL_ACTIVE
  Action: Decrypt credential envelope; store credential in HSM; schedule rotation;
          schedule idle timeout; log issuance event.

REQUESTING_CREDENTIAL --[APPROVAL_PENDING]--> APPROVAL_PENDING
  Action: Store approval_token; begin polling timer.

REQUESTING_CREDENTIAL --[CREDENTIAL_REJECTED]--> AID_ACTIVE
  Action: Log rejection; apply backoff if rate-limited; notify human principal if persistent.

APPROVAL_PENDING --[APPROVAL_GRANTED / clc_signatures_valid]--> CREDENTIAL_ACTIVE
  Action: Same as REQUESTING_CREDENTIAL -> CREDENTIAL_ACTIVE.

APPROVAL_PENDING --[APPROVAL_REJECTED]--> AID_ACTIVE
  Action: Log rejection and reason.

APPROVAL_PENDING --[APPROVAL_TIMEOUT / timeout_action_deny]--> AID_ACTIVE
  Action: Log timeout; record for compliance audit.

CREDENTIAL_ACTIVE --[ROTATION_TIMER_FIRED / clc_not_within_15pct_expiry]--> ROTATING
```

```
    Action: Generate new Ed25519 keypair; submit ARP rotation request to Hub.

  CREDENTIAL_ACTIVE --[ROTATION_TIMER_FIRED / clc_within_15pct_expiry]--> ROTATING
    Action: Generate new keypair; submit ARP; ALSO alert human principal of approach.

  CREDENTIAL_ACTIVE --[SCOPE_ESCALATION_NEEDED / scope_in_ceiling]--> ESCALATING
    Action: Construct escalation request; sign and submit to Hub.

  CREDENTIAL_ACTIVE --[DELEGATION_NEEDED / delegation_permitted]--> DELEGATING
    Action: Validate sub-scopes; construct delegation request; submit to Hub.

  CREDENTIAL_ACTIVE --[IDLE_TIMEOUT_FIRED]--> AID_ACTIVE
    Action: Mark CLC idle-expired locally; notify Hub; release HSM credential.

  CREDENTIAL_ACTIVE --[CLC_REVOKED_SIGNAL]--> AID_ACTIVE
    Action: Immediately discard HSM credential; log revocation event; notify principal.

  CREDENTIAL_ACTIVE --[ERS_SIGNAL_RECEIVED]--> SUSPENDED
    Action: Discard all HSM credentials; halt all outbound API calls; await instructions.

  ROTATING --[ROTATION_SUCCESSFUL / new_clc_valid]--> CREDENTIAL_ACTIVE
    Action: Atomic swap (two-phase); discard old private key; reschedule rotation.

  ROTATING --[ROTATION_FAILED / retry_count_lt_max]--> ROTATING
    Action: Exponential backoff; resubmit rotation request.

  ROTATING --[ROTATION_FAILED / retry_count_gte_max]--> CREDENTIAL_ACTIVE
    Action: Log persistent failure; alert human principal; continue with current credential.

  ESCALATING --[ESCALATION_APPROVED]--> CREDENTIAL_ACTIVE
    Action: Store updated CLC; log escalation event.

  ESCALATING --[ESCALATION_DENIED]--> CREDENTIAL_ACTIVE
    Action: Log denial; continue with current scopes.

  DELEGATING --[DELEGATION_CREATED]--> CREDENTIAL_ACTIVE
    Action: Store sub_clc reference; track for cascade revocation.

  DELEGATING --[DELEGATION_FAILED]--> CREDENTIAL_ACTIVE
    Action: Log failure reason; notify orchestrator logic.

  SUSPENDED --[REINSTATEMENT_SIGNAL / new_aid_valid]--> AID_ACTIVE
    Action: Re-register new AID; discard old AID; re-request all needed credentials.

  SUSPENDED --[DECOMMISSION_SIGNAL]--> DECOMMISSIONED
    Action: Purge all keys from HSM; archive audit records; notify principal.
```

14.2 Hub Request Processing State Machine

```
Hub Request Processing State Machine (per credential request)

States: REQUEST_RECEIVED, VALIDATING_HEADERS, VALIDATING_AID,
        VALIDATING_MANIFEST, EVALUATING_POLICY, FORWARDING_TO_SP,
        AWAITING_SP_RESPONSE, AWAITING_HUMAN_REVIEW, DELIVERING_CLC,
        REQUEST_COMPLETE, REQUEST_FAILED

Initial state: REQUEST_RECEIVED

Transitions:

REQUEST_RECEIVED --[HTTP_REQUEST_ARRIVES]--> VALIDATING_HEADERS
  Action: Record request receipt timestamp; assign internal tracking ID.

VALIDATING_HEADERS --[HEADERS_VALID / timestamp_ok AND request_id_unique]--> VALIDATING_AID
  Action: Store deduplication entry for request_id (TTL: 24 hours).

VALIDATING_HEADERS --[TIMESTAMP_OUT_OF_RANGE]--> REQUEST_FAILED
  Action: Return 400 TIMESTAMP_OUT_OF_RANGE; log.

VALIDATING_HEADERS --[DUPLICATE_REQUEST_ID]--> REQUEST_FAILED
  Action: Return 400 DUPLICATE_REQUEST_ID; log.

VALIDATING_AID --[AID_VALID / sig_ok AND not_expired AND not_revoked]--> VALIDATING_MANIFEST
  Action: Cache AID metadata for request lifetime.

VALIDATING_AID --[AID_EXPIRED]--> REQUEST_FAILED
  Action: Return 401 AID_EXPIRED.

VALIDATING_AID --[AID_REVOKED]--> REQUEST_FAILED
  Action: Return 401 AID_REVOKED; log revocation attempt.

VALIDATING_AID --[AID_SIGNATURE_INVALID]--> REQUEST_FAILED
  Action: Return 401 AID_SIGNATURE_INVALID; increment anomaly counter for source IP.

VALIDATING_MANIFEST --[MANIFEST_VALID / schema_ok AND sig_ok AND aid_matches]--> EVALUATING_POLICY
  Action: Resolve service_id; verify all requested_scopes are available.

VALIDATING_MANIFEST --[SCHEMA_INVALID]--> REQUEST_FAILED
  Action: Return 400 INVALID_MANIFEST with field-level errors.

VALIDATING_MANIFEST --[AGENT_SIG_INVALID]--> REQUEST_FAILED
```

```
    Action: Return 401 MANIFEST_SIGNATURE_INVALID; log.

  VALIDATING_MANIFEST --[SERVICE_NOT_FOUND]--> REQUEST_FAILED
    Action: Return 400 SERVICE_NOT_FOUND.

  EVALUATING_POLICY --[POLICY_APPROVE]--> FORWARDING_TO_SP
    Action: Record policy decision with matched rule_id in audit log.

  EVALUATING_POLICY --[POLICY_REVIEW]--> AWAITING_HUMAN_REVIEW
    Action: Create pending review record; send notification; return 202 to agent.

  EVALUATING_POLICY --[POLICY_REJECT]--> REQUEST_FAILED
    Action: Return 403 REQUEST_REJECTED with policy_rule identifier; log.

  FORWARDING_TO_SP --[SP_REQUEST_SENT]--> AWAITING_SP_RESPONSE
    Action: Record SP forwarding timestamp; start SP response timeout timer (30s).

  AWAITING_SP_RESPONSE --[SP_RESPONSE_RECEIVED / sp_sig_valid]--> DELIVERING_CLC
    Action: Verify SP signature; add hub_binding; sign hub_binding.

  AWAITING_SP_RESPONSE --[SP_TIMEOUT]--> REQUEST_FAILED
    Action: Return 503 SP_UNAVAILABLE; log; increment SP reliability counter.

  AWAITING_HUMAN_REVIEW --[HUMAN_APPROVED / reviewer_authorised]--> FORWARDING_TO_SP
    Action: Record approval with reviewer identity; log.

  AWAITING_HUMAN_REVIEW --[HUMAN_REJECTED]--> REQUEST_FAILED
    Action: Record rejection with reviewer identity and reason; log.

  AWAITING_HUMAN_REVIEW --[REVIEW_TIMEOUT / timeout_action_approve]--> FORWARDING_TO_SP
    Action: Log auto-approval on timeout; record timeout event.

  AWAITING_HUMAN_REVIEW --[REVIEW_TIMEOUT / timeout_action_reject]--> REQUEST_FAILED
    Action: Return rejection to polling agent; log.

  DELIVERING_CLC --[CLC_DELIVERED]--> REQUEST_COMPLETE
    Action: Record delivery timestamp; write audit ledger entry; start rotation schedule.
```

14.3 Service Provider CLC State Machine

```
Service Provider CLC State Machine (per issued CLC)

States: CREDENTIAL_PENDING, CREDENTIAL_ACTIVE, ROTATION_IN_PROGRESS,
        GRACE_PERIOD, REVOKED, EXPIRED

Initial state: CREDENTIAL_PENDING

Transitions:

CREDENTIAL_PENDING --[CLC_ISSUED / hub_sig_valid AND beneficiary_aid_known]--> CREDENTIAL_ACTIVE
  Action: Activate credential in SP credential store; start validity timer.

CREDENTIAL_ACTIVE --[API_CALL_WITH_CLC / clc_id_in_request]--> CREDENTIAL_ACTIVE
  Action: Validate credential (unexpired, unrevoked, scope matches resource);
          permit or deny API call; update last-used timestamp.

CREDENTIAL_ACTIVE --[ROTATION_REQUEST_FROM_HUB / hub_sig_valid]--> ROTATION_IN_PROGRESS
  Action: Record new public key; begin issuing replacement CLC.

CREDENTIAL_ACTIVE --[REVOCATION_SIGNAL_FROM_HUB / hub_sig_valid]--> REVOKED
  Action: Immediately mark credential invalid; reject all subsequent API calls.

CREDENTIAL_ACTIVE --[VALIDITY_EXPIRED]--> EXPIRED
  Action: Mark credential expired; reject all subsequent API calls.

ROTATION_IN_PROGRESS --[NEW_CLC_ISSUED]--> CREDENTIAL_ACTIVE (new CLC)
  Action: Activate new CLC; set old CLC to GRACE_PERIOD; return new CLC to Hub.

GRACE_PERIOD --[GRACE_PERIOD_ELAPSED]--> EXPIRED
  Action: Mark old credential expired.

GRACE_PERIOD --[API_CALL_WITH_OLD_CLC / within_overlap_window]--> GRACE_PERIOD
  Action: Accept call; log use of old credential during overlap.

GRACE_PERIOD --[API_CALL_WITH_OLD_CLC / outside_overlap_window]--> GRACE_PERIOD
  Action: Reject call; return 401 CLC_ROTATED.

REVOKED --[ANY_API_CALL]--> REVOKED
  Action: Reject; return 401 CLC_REVOKED; log attempt.

EXPIRED --[ANY_API_CALL]--> EXPIRED
  Action: Reject; return 401 CLC_EXPIRED; log attempt.
```

15. Extended Security Analysis

Formal Security Definitions

We use the standard game-based security framework throughout. A security game is a two-phase interaction between a challenger C and an adversary A. Security is defined as A's advantage being negligible in the security parameter lambda.

Definition 1 (EUF-CMA). An Ed25519 signature scheme Sigma is (t, epsilon)-EUF-CMA secure if for all PPT adversaries A running in time t, the advantage Adv_Sigma(A) of producing a valid signature on a new message is at most epsilon(lambda), where epsilon is a negligible function and the EUF-CMA game is the standard existential unforgeability under chosen-message attack game.

Definition 2 (IND-CCA2). An encryption scheme Enc is (t, epsilon)-IND-CCA2 secure if for all PPT adversaries A running in time t, the advantage of distinguishing encryptions in the standard IND-CCA2 game is at most epsilon(lambda).

Definition 3 (PRF). A function family F:{0,1}^k x {0,1}^* -> {0,1}^n is a (t, epsilon) -secure PRF if for all PPT distinguishers D running in time t, the advantage of distinguishing F from a truly random function is at most epsilon(lambda).

Definition 4 (CDH). The Computational Diffie-Hellman problem is (t, epsilon)-hard in group G if for all PPT algorithms A, Pr[A(g, g^a, g^b) = g^ab] <= epsilon(lambda) for random a, b in Z_p.

Assumption 1. Ed25519 is EUF-CMA secure under the discrete logarithm assumption on Curve25519.

Assumption 2. AES-256-GCM is IND-CCA2 secure with 256-bit keys and 96-bit nonces.

Assumption 3. HKDF-SHA256 is a secure PRF family per [Krawczyk10].

Assumption 4. ECDH on Curve25519 is CDH-hard.

Theorem 1 (AID Unforgeability). Under Assumption 1, no PPT adversary can produce a valid AID not previously registered by a certified IA, except with negligible probability.

Proof. An AID is valid iff ia_signature is a valid Ed25519 signature under the IA's registered public key. Forging a valid AID requires producing a signature under an IA private key without access to that key. By Assumption 1 (EUF-CMA of Ed25519), this succeeds with probability at most epsilon(lambda). The IA's private key is never transmitted in any AGEX protocol flow. QED.

Corollary 1 (Agent Authentication, SR-2). From Theorem 1 and the requirement that request signatures be verified against the AID public_key, an adversary cannot authenticate as an agent without the agent's private key.

Theorem 2 (Audit Ledger Tamper Evidence, SR-7). The AGEX Audit Ledger provides tamper-evident storage such that any modification to a historical event record is detectable by any party with access to the ledger.

Construction. Each event record e_i is stored with: h_i = SHA3-256(CANONICAL_JSON(e_i) || h_{i-1}) where h_0 is a published genesis hash.

Proof. Modifying e_j to e_j' gives h_j' != h_j with overwhelming probability (by SHA3-256 collision resistance: 2^{-256}). By induction, all subsequent hashes differ. Any verifier recomputing the chain will detect the inconsistency. QED.

Theorem 3 (APL Determinism). For any APL policy document P, credential request R, and evaluation context C, the Hub Policy Engine Eval(P, R, C) returns exactly one of {approve, review, reject} deterministically.

Proof. Rules are evaluated in strict priority order. Each condition type is a total boolean function of R and C with no undefined values. Evaluation terminates at the first matching rule's action, or returns default_action if no rule matches. Therefore Eval is a total deterministic function. QED.

Corollary 2 (APL Non-Repudiation). The policy document version is hashed and recorded in the Audit Ledger at each CLC issuance. Any post-hoc evaluation dispute can be resolved by re-running Eval with the archived policy version and request.

15.4 AHFP Security Properties

Security Analysis of the Hub Federation Protocol (AHFP)

AHFP-P1 (CHA Authenticity). A valid CHA can only be produced by Hub-A. Forging one requires Hub-A's private key, protected by Ed25519 EUF-CMA. Hub-B verifies the CHA signature and Hub-A's certificate against the Root of Trust.

AHFP-P2 (Trust Tier Integrity). Hub-B accepts the trust_tier asserted in the CHA. A compromised Hub-A could assert elevated tiers. Mitigations: IA-signed AIDs independently verify the trust tier; Hub operators are auditable by the Foundation; Root of Trust certification requires audit log maintenance.

AHFP-P3 (CHA Freshness). CHAs carry timestamp and valid_for_seconds. Hub-B MUST reject CHAs older than valid_for_seconds from the timestamp, preventing replay attacks.

AHFP-P4 (Bilateral Revocation Check). Both Hub-A (at attestation) and Hub-B (at CLC issuance) independently verify AID revocation. An AID revoked between these two checks is caught by Hub-B. Maximum exposure window equals revocation propagation latency.

AHFP-P5 (Hub Isolation). A compromised Hub-A cannot access credentials issued by Hub-B's service providers; credentials are encrypted under the beneficiary agent's public key (zero-knowledge, SR-1). The compromised Hub-A can misattest trust tiers but cannot issue Hub-B credentials.

16. Implementation Considerations

16.1 Performance

Performance Considerations

Credential Request Latency. For auto-approved requests (no human review), the dominant latency is SP forwarding (one HTTPS round-trip). Hub implementations SHOULD target p50 under 200ms and p99 under 1000ms end-to-end including TLS handshake.

Policy Evaluation. APL policies are on the hot path. Hub implementations SHOULD compile policy documents to efficient internal representations on load. Policy evaluation SHOULD complete in under 5ms at p99 for deployments exceeding 1000 requests/second.

AID Revocation Index. Consulted on every request. Implementations SHOULD maintain the index in memory for sub-millisecond lookup. For millions of AIDs, a Bloom or Cuckoo filter for fast non-membership checks is recommended, with exact lookup on positive hits.

Credential Envelope Encryption. ECDH-ES key agreement is performed by the SP. Hub computational overhead is dominated by Ed25519 signature verification, which runs at over 30,000 verifications/second on modern hardware.

ARP Rotation Coordination. High-concurrency rotation coordination SHOULD use an event-driven architecture rather than thread-per-request. Idempotent SP endpoint design enables safe retries without duplicate CLC issuance.

16.2 High Availability and Disaster Recovery

High Availability and Disaster Recovery

Hub Availability Targets. The Identity Registry and Rotation Coordinator MUST maintain 99.99% uptime (HC-M-008). This requires multi-region active-active deployment with synchronous replication of the revocation index and asynchronous replication of audit records.

Database Replication. AID Registry, CLC Store, Policy Store, and Pending Approval Store require synchronous multi-master replication across at least two availability zones. The Audit Ledger requires write-ordering guarantees; a single-writer, multi-reader design prevents hash chain forking.

ERS Availability. The ERS subsystem SHOULD be deployed with independent redundancy from the credential brokering layer. ERS signals SHOULD be durably queued (e.g. Kafka or NATS JetStream) to survive Hub process failures.

SP Unavailability During Rotation. When an SP is unavailable during ARP, the Hub SHOULD hold the request in a durable retry queue with exponential backoff up to 30 minutes. If the SP remains unavailable past CLC validity, the Hub MUST notify the agent and human principal.

Recovery Objectives. Recovery Time Objective (RTO): 4 hours from total Hub failure. Recovery Point Objective (RPO): 60 seconds for the Audit Ledger; 5 minutes for the AID Registry and CLC Store.

16.3 Key Management

Key Management Guidance

Hub Private Key. Hub operators MUST store the Hub private key in an HSM validated to FIPS 140-3 Level 3 or equivalent (CC EAL 4+). The HSM MUST enforce usage policies restricting the key to AGEX Ed25519 signing operations only. Dual-control HSM activation SHOULD be enforced for Tier 1+ Hub operators.

Agent Private Keys. Agent private keys SHOULD be stored in an HSM. For cloud deployments without physical HSM access, cloud-provider KMS services (AWS KMS, Google Cloud KMS, Azure Key Vault) are acceptable for Tier 0 and Tier 1 agents. Tier 3 agents in regulated environments MUST use FIPS 140-3 validated key storage.

Key Generation Entropy. Ed25519 key generation requires 32 bytes of cryptographically secure random data from the OS CSPRNG (/dev/urandom on Linux, CryptGenRandom on Windows). Application-level PRNGs MUST NOT be used for key material. In virtualised environments, entropy source seeding SHOULD be verified at startup.

Hub Key Rotation Ceremonies. Hub private keys MUST be rotated at least annually. Ceremonies MUST be documented, dual-controlled, and audited. The new Hub public key MUST be registered with the Root of Trust before the old key is retired. A 30-day transition period MUST be observed during which both keys are accepted.

IA Key Management. IA private keys MUST be stored in offline HSMs not connected to any network. AID signing ceremonies MUST be performed in an air-gapped environment. Online signing SHOULD use sub-keys with limited validity periods derived from the offline root key.

16.4 Monitoring and Alerting

```
Operational Monitoring and Alerting

Hub operators MUST implement monitoring for the following metrics and alert when thresholds
are exceeded.

Request Rate Metrics:
  credential_requests_per_minute  -- Alert: baseline + 3 standard deviations
  approval_rate                   -- Alert: below 80% without policy change
  rejection_rate                  -- Alert: above 5% for any service
  sp_error_rate                   -- Alert: above 1% for any service

Performance Metrics:
  credential_request_latency_p99  -- Alert: above 2 seconds
  arp_rotation_latency_p99        -- Alert: above 10 seconds
  ers_completion_time_ms          -- Alert (warn): above 30 seconds; (critical): above 60 seconds
  policy_evaluation_latency_p99   -- Alert: above 50ms

Security Metrics:
  aid_signature_failures_per_hour        -- Alert: above 10 (potential probing)
  duplicate_request_id_count             -- Alert: any non-zero (potential replay)
  timestamp_out_of_range_count           -- Alert: any non-zero (potential time-skew attack)
  ers_signals_per_hour                   -- Alert security team: above 5
  failed_rotation_count_per_clc          -- Alert principal: above 3 for any single CLC

Audit Ledger Integrity:
  audit_chain_verification  -- Run every 15 minutes; alert immediately on any failure
  audit_write_latency_p99   -- Alert: above 5 seconds
```

16.5 Rate Limiting

```
Rate Limiting Specification

Per-AID Rate Limits:
  10 credential requests per minute
   1 rotation request per minute per CLC
   5 escalation requests per hour per CLC
  20 delegation requests per hour

Per-Organisation Rate Limits:
  100 credential requests per minute per org_id
   10 ERS signals per hour per org_id (override requires Hub operator approval)

Per-Source-IP Rate Limits:
   50 requests per minute per source IP
  500 requests per hour per source IP

Rate Limit Response (HTTP 429):

  HTTP/1.1 429 Too Many Requests
  Retry-After: <seconds>
  X-RateLimit-Limit: <limit>
  X-RateLimit-Remaining: 0
  X-RateLimit-Reset: <unix-timestamp>

  {
    "error": {
      "code":        "RATE_LIMIT_EXCEEDED",
      "message":     "Rate limit exceeded for this AID/organisation/IP",
      "retry_after": <seconds>,
      "limit_type":  "<aid|org|ip|global>",
      "request_id":  "<uuid-v4>"
    }
  }

Agents MUST honour the Retry-After header and implement exponential backoff.
Agents MUST NOT implement tight retry loops ignoring rate limit responses.
```

16.6 Versioning and Backwards Compatibility

Protocol Versioning and Backwards Compatibility

AGEX uses MAJOR.MINOR semantic versioning. The version is expressed in the X-AGEX-Version request header and X-AGEX-Server-Version response header.

Backwards Compatibility Policy. Within a major version, the Hub MUST accept all minor versions. Unknown optional fields MUST be ignored. Removing a field or changing its semantics requires a major version increment.

Deprecation Policy. Fields may be marked deprecated with a minimum 12-month deprecation period before removal. Deprecated fields appear in the Hub's federation metadata document under a deprecated_fields array.

Major Version Transitions. Both versions MUST be simultaneously supported for a minimum 6-month transition period. Compatibility shim documentation is published for each major version transition. Agents may transition at their own pace within the supported period.

17. Governance and Implementation Roadmap

14.1 The AGEX Foundation

The AGEX Foundation is an independent, non-profit organisation established to steward the AGEX protocol specification, certification programme, and community governance. The Foundation is vendor-neutral and structured to prevent any single organisation from controlling the protocol's direction. Three governing bodies operate the Foundation: the Technical Steering Committee (9 elected seats, 2-year terms, no organisation holds more than 2 seats), the Security Advisory Board (cryptographic and threat model oversight), and the Certification Authority (Hub operator and service provider certification).

14.2 Intellectual Property Policy

The AGEX protocol specification is published under Creative Commons Attribution 4.0 International (CC BY 4.0). Any party may implement, extend, or commercialise the specification without royalty or restriction. All contributors sign the AGEX Foundation Contributor Licence Agreement. The AGEX trademark requires written Foundation permission for use in product names, granted freely to conformant implementations.

14.3 IETF Standards Path

The Foundation pursues standardisation through the IETF in four phases: Phase 1 (Q1-Q2 2026) - informational RFC submission establishing the problem statement and solution; Phase 2 (Q2-Q3 2026) - IETF Working Group formation with chartered deliverables; Phase 3 (2026-2027) - Proposed Standard RFC for AGEX Core Protocol; Phase 4 (2027+) - Internet Standard following two or more interoperable independent implementations.

14.4 Implementation Roadmap

Phase	Period	Deliverable
1	Q1-Q2 2026	Full specification v0.9, conformance test suite v0.1, Foundation entity, TSC and SAB established
2	Q2-Q3 2026	Open source Hub reference implementation (Apache 2.0), Python and Node.js SDKs, developer sandbox, AGEX Certified badge programme
3	Q3-Q4 2026	Agent framework integrations (LangChain, AutoGen, CrewAI, n8n), API provider onboarding, IETF informational RFC submission
4	2027	AGEX v1.0 stable spec, IETF Working Group, enterprise Hub certification, first AGEX Security Summit

18. Conclusion

The absence of an authentication protocol designed for autonomous AI agents is not a gap that can be resolved through creative application of existing standards. It is a structural incompatibility between protocols designed for human-delegated interaction and a deployment context with fundamentally different requirements. As agent deployments scale from thousands to millions of concurrent autonomous processes, the credential management crisis will become the dominant security challenge of enterprise computing.

AGEX proposes to resolve this crisis with a complete, open, and extensible protocol for the full lifecycle of agent credentials. The five AGEX primitives, AID, Intent Manifest, CLC, ARP, and Delegation Chains, together provide a trust infrastructure that is cryptographically sound under standard assumptions, operationally practical under real deployment constraints, and extensible to the full diversity of agent types and service provider environments.

The formal security analysis demonstrates that AGEX satisfies all eight security requirements defined in Section 3.3. The comparative evaluation demonstrates that no existing protocol addresses more than 8 of the 12 criteria relevant to autonomous agent operation, and that AGEX addresses all 12. The six worked deployment scenarios demonstrate practical applicability across consumer, enterprise, multi-agent, cross-organisational, compliance-sensitive, and incident-response contexts.

The standardisation window is narrow. The AGEX Foundation invites the community to review the specification, implement the protocol, contribute to the open source reference implementation, and participate in the Foundation's governance process. The agentic web is being built now. AGEX is the credential infrastructure it requires.

> *"OAuth2 was built for humans. The agentic web needs a protocol built for agents. AGEX is that protocol."*

References

[RFC2119] S. Bradner, Key words for use in RFCs to Indicate Requirement Levels, IETF RFC 2119, March 1997.

[RFC4122] P. Leach et al., A Universally Unique IDentifier (UUID) URN Namespace, IETF RFC 4122, July 2005.

[RFC6749] D. Hardt (Ed.), The OAuth 2.0 Authorization Framework, IETF RFC 6749, October 2012.

[RFC7009] T. Lodderstedt et al., OAuth 2.0 Token Revocation, IETF RFC 7009, August 2013.

[RFC7523] M. Jones et al., JSON Web Token (JWT) Profile for OAuth 2.0 Client Authentication, IETF RFC 7523, May 2015.

[RFC7591] J. Richer (Ed.), OAuth 2.0 Dynamic Client Registration Protocol, IETF RFC 7591, July 2015.

[RFC7617] J. Reschke, The Basic HTTP Authentication Scheme, IETF RFC 7617, September 2015.

[RFC7636] N. Sakimura et al., Proof Key for Code Exchange by OAuth Public Clients, IETF RFC 7636, September 2015.

[RFC8032] S. Josefsson, I. Liusvaara, Edwards-Curve Digital Signature Algorithm (EdDSA), IETF RFC 8032, January 2017.

[RFC8037] I. Liusvaara, CFRG Elliptic Curves for JOSE, IETF RFC 8037, January 2017.

[RFC8414] M. Jones et al., OAuth 2.0 Authorization Server Metadata, IETF RFC 8414, June 2018.

[RFC8446] E. Rescorla, The Transport Layer Security (TLS) Protocol Version 1.3, IETF RFC 8446, August 2018.

[RFC8471] T. Attema et al., The Token Binding Protocol Version 1.0, IETF RFC 8471, October 2018.

[RFC8259] T. Bray (Ed.), The JavaScript Object Notation (JSON) Data Interchange Format, IETF RFC 8259, December 2017.

[RFC8628] B. Campbell et al., OAuth 2.0 Device Authorization Grant, IETF RFC 8628, August 2019.

[RFC8705] B. Campbell et al., OAuth 2.0 Mutual-TLS Client Authentication, IETF RFC 8705, February 2020.

[RFC8785] A. Rundgren et al., JSON Canonicalization Scheme (JCS), IETF RFC 8785, June 2020.

[RFC9126] T. Lodderstedt et al., OAuth 2.0 Pushed Authorization Requests, IETF RFC 9126, September 2021.

[RFC9396] T. Lodderstedt et al., OAuth 2.0 Rich Authorization Requests, IETF RFC 9396, May 2023.

[SPIFFE] E. Ace et al., Secure Production Identity Framework for Everyone (SPIFFE), CNCF Specification v1.0, 2021.

[W3C-DID] M. Sporny et al., Decentralized Identifiers (DIDs) v1.0, W3C Recommendation, July 2022.

[W3C-VC] M. Sporny et al., Verifiable Credentials Data Model v1.1, W3C Recommendation, March 2022.

[NIST-AI] NIST, Artificial Intelligence Risk Management Framework (AI RMF 1.0), NIST AI 100-1, January 2023.

[EU-AIA] European Parliament, Regulation (EU) 2024/1689 (Artificial Intelligence Act), Official Journal of the EU, July 2024.

[OWASP-API] OWASP Foundation, OWASP API Security Top 10, 2023 Edition.

[Bernstein06] D.J. Bernstein, Curve25519: New Diffie-Hellman Speed Records, Public Key Cryptography (PKC) 2006, LNCS 3958, pp. 207-228.

[NIST-GCM] M. Dworkin, Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM), NIST SP 800-38D, November 2007.

[Krawczyk10] H. Krawczyk, Cryptographic Extraction and Key Derivation: The HKDF Scheme, CRYPTO 2010, LNCS 6223.

[GDPR] European Parliament, Regulation (EU) 2016/679 (General Data Protection Regulation), Official Journal of the EU, May 2018.

[CCPA] California Consumer Privacy Act of 2018, Cal. Civ. Code Section 1798.100 et seq., as amended by CPRA 2020.

15a. IETF Standards Track Considerations

This section addresses considerations specific to IETF Standards Track review that are relevant to implementers and community reviewers engaging with the AGEX specification. These considerations follow the structure recommended by the IETF Area Director review guidelines and address questions routinely raised in IETF Last Call and IESG review processes.

Protocol Design Rationale and Alternatives Considered

The AGEX specification team considered several alternative designs before arriving at the current architecture. This section documents those alternatives and the reasoning behind the choices made, preserving institutional knowledge for the IETF Working Group deliberation process.

AID Format: JSON vs X.509

The AGEX AID uses a JSON structure with an Ed25519 signature rather than an X.509 certificate. The X.509 alternative was considered extensively. X.509 certificates offer mature tooling, widespread deployment experience, and the established PKI infrastructure that many organisations already operate. However, four factors favoured the JSON approach for the agent identity use case.

First, the AI agent developer community is web-native and JSON-fluent; certificate handling APIs are complex by comparison and create a higher barrier to correct implementation. Second, AGEX's agent-specific metadata fields (capabilities, trust_tier, restrictions) would require X.509 extensions, introducing Additional X.509 Extension complexity that does not exist for JSON. Third, certificate revocation in X.509 (CRL and OCSP) has well-documented deployment challenges; AGEX's AID revocation model integrates directly with the Hub's revocation index without requiring separate infrastructure. Fourth, the DID ecosystem (W3C DID Core) provides a compatible identity format for AGEX to interoperate with when the deployment context warrants it.

The security properties achievable with the JSON + Ed25519 approach are equivalent to X.509 + ECDSA-P256 for the relevant threat model. The tradeoff is that the AGEX trust hierarchy is defined by the specification rather than inherited from the existing Web PKI.

Credential Envelope: JWE vs Custom Encryption

The credential_envelope uses JWE (RFC 7516) with ECDH-ES+AES256GCM. Alternatives considered include: a custom binary envelope format (rejected for implementability reasons; JSON with standard JOSE tooling is preferred), pure transport security without application-layer envelope encryption (rejected because it would not achieve the Hub zero-knowledge property SR-8), and RSA-OAEP (rejected because RSA key sizes required for equivalent security are substantially larger than Curve25519 keys, and RSA lacks perfect forward secrecy in this construction).

The JWE approach provides two key advantages: it uses well-specified, widely-audited cryptographic constructions from the JOSE family, and the protected header carries algorithm metadata enabling future algorithm agility without protocol version changes.

Hub Architecture: Centralised vs Peer-to-Peer

AGEX adopts a Hub mediator architecture rather than a peer-to-peer credential exchange. In a P2P design, agents would negotiate credentials directly with service providers, with no central mediator. This was considered and rejected for three reasons: (1) it requires every service provider to independently implement the full verification, policy evaluation, and lifecycle management stack, creating a high adoption barrier; (2) it makes cross-service audit aggregation and emergency revocation significantly harder without a coordinating entity; (3) the AHFP federation protocol preserves decentralisation at the organisational level while maintaining operational simplicity.

The Hub is not a trust anchor for credentials; the service provider remains the issuant authority. The Hub is a mediator and lifecycle coordinator. This distinction is architecturally significant: a compromised Hub cannot forge credentials (SR-2) because it does not hold the SP's signing key.

Trust Tier Granularity

The four-tier (0-3) trust model was chosen over alternatives including: continuous trust scores (0.0-1.0), capability-based access lists, and binary trusted/untrusted classification. The discrete tier model was preferred because it aligns with how security teams categorise risk in practice, makes APL policy authorship more intuitive (tier >= 2 is a natural policy predicate), and creates clear graduation criteria that can be audited by certification bodies.

Open Design Issues for WG Deliberation

The following issues are formally raised for resolution through the IETF Working Group process. Each issue is stated as a design question with the current specification's answer and the alternative approach that has been proposed.

Issue 1 — Hub Concentration Risk: The current AHFP specification enables multi-Hub interoperability, but in practice many deployments will use a small number of major Hub operators, creating concentration risk. Should the specification include technical mechanisms (such as mandatory multi-Hub redundancy for Tier 2/3 agents) to prevent over-concentration? Current answer: No mandatory redundancy; the AGEX Foundation certification programme addresses this through Hub operator terms of service.

Issue 2 — Revocation Propagation Bound: The 60-second ERS completion bound is appropriate for intra-region deployments but may be difficult to meet for cross-region or cross-continental service provider deployments with high network latency. Should the bound be relaxed for explicitly declared cross-region CLCs? Current answer: The 60-second bound applies uniformly; Hub operators SHOULD deploy regionally to meet it.

Issue 3 — AID Self-Renewal: The current specification requires manual interaction with the issuing IA to renew an expiring AID, analogous to certificate renewal in PKI. Should an automated AID renewal protocol (analogous to ACME for X.509 certificates) be standardised as part of AGEX 1.0? Current answer: AID renewal is deferred to a companion specification.

Issue 4 — Scope Name Standardisation: Service providers define their own scope vocabulary, which limits portability of APL policies across providers. Should AGEX define a standard scope vocabulary for common resource types? Current answer: A scope registry will be proposed to IANA as part of the IETF standardisation process.

Issue 5 — CLC Freshness vs Rotation: The ARP rotation mechanism is the only way to obtain fresh credentials under the current specification. A lightweight freshness token mechanism (with lower overhead than full ARP rotation) could be useful for short-lived CLCs. Current answer: This use case is addressed by setting a short rotation_interval_seconds; a dedicated freshness mechanism is not specified.

IANA Considerations

Upon IETF standardisation, the AGEX specification requires the following IANA actions.

A new Media Type registration for application/agex+json is required for AGEX protocol messages that are referenced from outside the HTTP API context (for example, in email-transmitted approval notifications or in document storage of AID records).

A new AGEX Parameters Registry is required for the following extensible fields: rotation_reason values (currently: scheduled, compromise_suspected, policy_change, principal_request), trust_tier values, credential_envelope algorithm identifiers, and AID capability string values.

A new AGEX Error Codes Registry is required for the error code strings used in all Hub API error responses, to enable interoperability verification between independent implementations.

A new Well-Known URI registration (/.well-known/agex-hub) is required for Hub discovery, analogous to the OIDC discovery document at /.well-known/openid-configuration.

15b. Extended Comparative Analysis

This section expands on the twelve-criterion comparative evaluation presented in Section 10, providing detailed analysis for each criterion and examining specific technical limitations that explain the scoring assigned to each protocol.

Criterion 1: Agent-Native Identity Primitives

OAuth 2.0 assigns a client_id that is typically a human-readable string assigned by the authorisation server at client registration time (RFC 7591). This identifier has no cryptographic binding: the client authenticates using a shared secret (client_secret) or, under RFC 7523, a pre-registered public key. The identity primitive carries no metadata about the client's nature, capabilities, or the organisational context in which it operates. Crucially, there is no standard mechanism for a client_id to declare that it represents an AI agent, as opposed to a web application or mobile app. Policy decisions about agent identity must therefore be made out-of-band, rather than being readable from the identity primitive itself.

OpenID Connect inherits OAuth 2.0's identity primitives and adds the ID token, a JWT assertion about a human user's identity. The ID token claim set (sub, name, email, phone_number, birthdate, etc.) has no meaningful mapping to an autonomous agent, and OIDC provides no standard way to issue identity assertions for non-human actors.

SPIFFE provides a significant improvement with the SPIFFE ID, a URI that encodes the workload's identity within a trust domain (spiffe://trust-domain/path). The SPIFFE SVID carries cryptographic binding through the certificate or JWT format. However, the SPIFFE ID carries no agent-specific metadata: it identifies a workload but says nothing about the workload's capabilities, the human principal on whose behalf it operates, or its trust history. SPIFFE partial credit reflects its cryptographic identity binding without agent-specific semantic enrichment.

AGEX's AID provides cryptographically bound agent identity with full semantic enrichment: agent type, capabilities, owning organisation, trust tier (determined by an independent Identity Authority, not self-asserted), and operational restrictions. The AID is designed specifically for the agent identity use case.

Criterion 4: Autonomous Credential Rotation Without Downtime

OAuth 2.0 access token rotation occurs implicitly via the token lifetime: tokens expire and the client requests a new one. Client credential rotation (the underlying client_secret) has no standard mechanism. RFC 7591 allows programmatic client registration but not client credential rotation. In practice, organisations rotate OAuth 2.0 client secrets via manual processes with a maintenance window during which the old secret is still active. This is operationally acceptable for human-managed applications but fails for autonomous agents running continuous workloads.

SPIFFE provides the closest existing analogue to ARP. The SPIRE agent automatically renews SVIDs before they expire, typically when 50% of the SVID lifetime has elapsed. This is a form of autonomous rotation within a single trust domain. However, SPIFFE rotation is specific to X.509 certificates and JWTs; it does not address the rotation of API keys, OAuth tokens, or other credential types that agents commonly use. AGEX ARP is protocol-agnostic: the credential_envelope can carry any credential type.

Criterion 8: Emergency Cascade Revocation

No existing standard provides emergency cascade revocation across multiple services. OAuth 2.0 token revocation (RFC 7009 [RFC7009]) revokes a specific token at a specific authorisation server. This is one-token, one-server revocation. A compromised agent holding 20 CLCs from 15 different service providers requires 20 separate revocation calls, each to a different endpoint, with no guarantee that all revocations complete within any bounded time.

SPIFFE certificate revocation via CRL or OCSP revokes specific certificates at specific CAs. In a multi-CA SPIFFE deployment, cross-CA revocation still requires per-CA action. There is no SPIFFE primitive for "revoke all credentials associated with this workload across all trust domains it participates in."

AGEX ERS provides this capability as a first-class protocol operation. A single API call to the Hub initiates cascade revocation across all services, with a 60-second completion bound, an audit record of the cascade, and acknowledgment tracking for SPs that do not respond.

Criterion 10: Zero-Knowledge Credential Brokering

This criterion has no precedent in existing protocols because it addresses a threat specific to the Hub architecture. An OAuth 2.0 authorisation server holds the resource owner's authorisation decisions and the client's tokens; it is by design a party that knows both sides of the credential relationship. In AGEX, the Hub mediates the credential exchange without holding the credential plaintext.

The zero-knowledge property is achieved by the ECDH-ES envelope: the Hub receives the CLC from the SP with the credential already encrypted for the agent's public key, and delivers it without decrypting it. A compromised Hub operator cannot read any agent's credential material, even with full database access. This property has no equivalent in any existing authentication protocol because existing protocols were not designed with the Hub threat model in mind.

15c. Additional Security Theorems

This section presents additional security analysis supplementing the five theorems in Section 9. These results address properties not captured by the primary theorem set.

Theorem 6: Intent Integrity (SR-6)

Statement: If the agent_signature in the Intent Manifest is verified by the service provider, it is computationally infeasible to produce a CLC that references a manifest with different intent fields from those the agent declared.

Proof sketch: The CLC contains manifest_id and manifest_hash fields. manifest_hash is SHA-3-256 of the canonical JSON of the complete manifest. The manifest's canonical JSON includes all intent fields and is signed by the agent. A service provider receiving a credential request verifies manifest.agent_signature against the agent's AID public key. Modifying any manifest field after signing requires either: (a) finding a SHA-3-256 collision to produce a different manifest with the same hash (assumed infeasible under the hash collision resistance assumption), or (b) forging the agent's Ed25519 signature over the modified manifest (assumed infeasible under EUF-CMA). Therefore SR-6 holds under these assumptions. Note: this property requires SPs to verify agent_signature in the manifest, which is a normative requirement (SC-M-001 via hub-forwarding verification).

Theorem 7: Audit Non-Repudiation (SR-7)

Statement: An agent cannot credibly deny having requested or used a credential if the corresponding Audit Ledger events exist and the Ledger's hash chain integrity is intact.

Proof sketch: Each credential request carries X-AGEX-Signature, an Ed25519 signature over the request body, timestamp, and request_id, computed using the agent's private key. The Hub records this signature in the Audit Ledger event for the request. An agent cannot produce a valid signature without its private key (EUF-CMA). An adversary cannot modify the Audit Ledger event without invalidating the hash chain from that event to the current head. Therefore, the existence of an Audit Ledger event with a valid agent signature constitutes non-repudiable evidence of the agent's request.

Limitation: If the agent's private key is compromised before the audit event is recorded, an adversary could produce a signed request that is attributed to the agent. This is an inherent limitation of signature-based non-repudiation. The limitation is mitigated by HSM-based key storage (implementation guidance, Section 8.4) which prevents key extraction.

Theorem 8: Hub Zero-Knowledge (SR-8)

Statement: A Hub operator with full read access to all Hub storage (CLC database, audit log, in-flight request cache) cannot recover the credential plaintext for any agent.

Proof sketch: All credential plaintext is contained in credential_envelope fields within CLCs. The credential_envelope is encrypted using ECDH-ES: CEK = HKDF(ECDH(ephemeral_sk, agent_pk)), and the credential is AES-256-GCM encrypted under the CEK. The ephemeral secret key ephemeral_sk is generated by the service provider and discarded after encryption; it does not appear in any Hub-stored field. The Hub stores only: the JWE protected header (algorithm parameters, not key material), the epk (ephemeral public key), the ciphertext, the nonce, and the authentication tag. Decryption requires agent_sk, which is never transmitted to the Hub. Therefore, Hub database access yields no information about credential plaintext under the CDH and IND-CCA2 assumptions.

Corollary: A Hub operator cannot perform selective denial of service by issuing fake CLCs (CLCs containing invalid ciphertext), because the service provider signs the credential_envelope's JWE protected header as part of provider_signature. An invalid ciphertext would produce an authentication tag mismatch detectable at decryption time, identifiable as either Hub corruption or SP misbehaviour.

Security of the APL Policy Engine (Non-Interference)

The APL Policy Engine evaluates policies over credential request data. A potential security concern is that a maliciously crafted credential request could exploit the policy engine's evaluation logic to cause incorrect approval or denial. We analyse the policy engine's attack surface.

The policy engine operates on: the AID (verified signature, fixed schema), the Intent Manifest (verified agent signature, fixed schema), and the APL policy document (authored by the service provider, stored by the Hub). All three inputs are schema-validated before reaching the policy engine. The policy engine implements a pure boolean expression evaluator with no dynamic code execution, no network access, and no side effects. The evaluation context (the AID and manifest data) is immutable during evaluation.

A maliciously crafted AID or manifest cannot cause the policy engine to execute arbitrary code because the engine's condition evaluation is implemented as structural pattern matching over typed JSON data, not string evaluation or dynamic dispatch. The policy engine's only output is a discrete action value (approve, review, or reject); it has no mechanism to return partial results or trigger secondary operations.

The policy engine is therefore not exploitable via crafted inputs under normal implementation assumptions. The primary risk to the policy engine's correctness is incorrect policy authorship (the service provider creating a policy with logical errors), which is addressed by the policy testing harness described in Section 8.3.

15d. Service Provider Integration Guide

This section provides a detailed integration guide for service providers implementing AGEX credential acceptance. The guide is organised by integration phase and covers the complete implementation path from initial setup to production operation.

Phase 1: Identity and Registration

A service provider begins AGEX integration by registering with the AGEX Foundation. Registration requires: legal entity documentation, contact information for the security team, and agreement to the AGEX Service Provider Terms of Service. The Foundation assigns the SP an sp_id (a UUID that serves as the persistent identifier for the service provider across all Hub registrations) and issues an SP certificate.

The SP selects one or more AGEX Hubs with which to register. Hub selection criteria include geographic coverage (match your user base's location), latency (test Hub response times from your API infrastructure), Hub tier (enterprise vs community Hub), and AHFP coverage (which other Hubs the selected Hub federates with). SP registration with a Hub is performed via the Hub's SP onboarding API, which is separate from the agent-facing AGEX API.

Phase 2: APL Policy Design

Before accepting any credential requests, the SP must design and deploy its APL policy. This is the most critical implementation phase for security correctness.

The policy design process SHOULD begin with a threat model: what are the worst-case credential abuses this policy must prevent? Common requirements include: preventing Tier 0 agents from obtaining write permissions (any agent with write access must be at least Tier 1), preventing PII-processing credentials from being issued to agents that do not declare pii_processing: true (to ensure GDPR Article 30 record-keeping), and preventing cross-border credentials from being issued for data with geographic restrictions.

The policy MUST have a default_action of reject. A policy with default_action: approve would approve any request not explicitly matched by a rule, creating a wide-open approval posture that defeats the purpose of the APL evaluation layer.

Rules SHOULD be numbered with a priority that reflects their importance. Security denials (data_classification: restricted, tier0 admin requests) SHOULD have the lowest priority numbers (evaluated first). Broad approval rules for well-understood cases SHOULD have higher priority numbers (evaluated last, only if no security denial matched).

The APL testing harness at test.agex.api allows SPs to test their policies against a corpus of 500 synthetic credential requests covering all trust tier combinations, all standard scope combinations, all intent types, and edge cases including PII processing, restricted data classification, and administrative scope requests. SPs SHOULD achieve 100% correct classification (expected action matches actual action) on the standard test corpus before deploying to production.

Phase 3: CLC Issuance Implementation

The SP implements a CLC issuance endpoint that the Hub calls when forwarding approved credential requests. This endpoint must: receive the forwarded request from the Hub, verify the Hub's signature on the forwarded request, generate the API credential to be enclosed in the CLC, construct the CLC object with all required fields, encrypt the credential in the credential_envelope, sign the CLC with the SP's private key, and return the signed CLC to the Hub.

The CLC's rotation_policy SHOULD be set based on the SP's risk appetite. High-security APIs (payment processing, health data, administrative operations) SHOULD use short rotation intervals (3600 seconds, 1 hour) to limit the exposure window of any compromised credential. Lower-risk APIs MAY use longer rotation intervals (86400 seconds, 24 hours) to reduce the volume of ARP rotation traffic. The rotation_overlap_seconds MUST be set to at least 60 seconds to ensure the agent has time to complete the swap before the old credential expires; 300 seconds (5 minutes) is recommended for most deployments.

The CLC's scope_ceiling SHOULD be set to the maximum scope the SP is willing to grant this agent via escalation. The scope_ceiling enables dynamic scope expansion within a pre-authorised bound without requiring a full new credential request. For Tier 0 agents, the scope_ceiling SHOULD be identical to granted_scopes (no escalation permitted). For Tier 3 agents with established trust, the scope_ceiling MAY extend significantly beyond the initially granted scopes.

Phase 4: Lifecycle Support

The SP must implement three inbound signal handlers from the Hub:

The ARP rotation signal informs the SP that an agent is rotating its credentials. The SP must revoke the old credential and issue a new credential encrypted under the agent's new public key, returned in a new CLC. The SP MUST complete this within 30 seconds. The SP SHOULD implement idempotency on the rotation endpoint: if the same rotation request_id is received twice (due to Hub retry), the SP MUST return the same new CLC it issued on the first call, not issue a second new credential.

The ERS revocation signal instructs the SP to immediately revoke all credentials associated with the specified CLC IDs or AID. The SP MUST complete revocation within 5 seconds of receiving the signal for per-CLC revocation, and within 10 seconds for AID-level revocation (which may affect multiple CLCs). The SP MUST return an acknowledgment to the Hub upon successful revocation; failure to acknowledge causes the Hub to continue retrying the revocation signal.

The suspension signal is similar to revocation but temporary. The SP marks the CLC as suspended; authenticated API calls using the suspended credential MUST be rejected with HTTP 403. A subsequent resume signal restores the CLC to active status.

Phase 5: Monitoring and Operations

In production, the SP SHOULD monitor: the rate of credential requests by trust tier (unusual spikes may indicate bot-driven enumeration), the rotation success rate (a high rate of rotation failures may indicate SP-side issues), the ERS signal receipt rate (unexpected ERS signals may indicate a security incident), and the audit report query rate (unusual query volumes may indicate a breach investigation).

SPs SHOULD implement webhook notifications to their security teams for: any Tier 0 agent credential request for write or admin scopes (these should always go through human review; an auto-approval is a policy misconfiguration), any ERS signal received for an AID that holds more than 10 CLCs (a large-scale revocation event warrants immediate investigation), and any APL policy evaluation that results in a rejection for a Tier 3 agent (Tier 3 agents should rarely be rejected under a correctly authored policy).

15e. Agent SDK Interface Reference

This section specifies the interface that AGEX agent SDKs SHOULD expose to agent framework developers. The interface is defined in protocol-agnostic pseudocode; specific SDK implementations will adapt it to the conventions of their target language (Python, TypeScript, Go, Java, etc.).

The AGEX Agent SDK provides three layers of abstraction: a low-level protocol layer (direct access to AGEX HTTP API methods), a lifecycle management layer (handles ARP rotation, expiry monitoring, and error recovery automatically), and a high-level task API (the simplest interface for agent framework integration).

High-Level Task API

The high-level task API is designed for agent framework integration where the developer wants minimal interaction with AGEX internals. It exposes three operations:

credentials = agex.use(service_id, scopes, intent, duration)

The use() call acquires credentials for the specified service, handling the full acquisition flow including ARP rotation and error recovery transparently. It blocks until credentials are available (or raises if acquisition fails). The returned credentials object exposes a single property, credentials.value, containing the decrypted API key or token. The SDK automatically refreshes credentials before expiry; the agent code need not manage rotation.

agex.release(service_id)

The release() call initiates voluntary CLC revocation for the specified service. The SDK sends the ERS self-revocation signal and deletes credential material from secure storage. Agent code SHOULD call release() when the task is complete rather than waiting for idle_timeout to fire, to minimise the credential exposure window.

with agex.task(service_id, scopes, intent, duration) as credentials: # agent code using credentials.value ...

The task context manager (available in Python-style SDKs) combines use() and release() into a single context block, ensuring release() is called even if the agent code raises an exception. This is the recommended pattern for agent framework integration.

Lifecycle Management Layer

The lifecycle management layer exposes the full credential lifecycle for use cases where the high-level task API is insufficient (for example, multi-service workflows where credential acquisition and release are decoupled from the code that uses them).

clc = agex.acquire(service_id, intent_manifest) credential = agex.get_credential(service_id) agex.rotate(service_id) agex.escalate(service_id, additional_scopes, justification) sub_clc = agex.delegate(service_id, child_aid, sub_scopes, duration) agex.revoke(service_id) agex.revoke_all() # ERS self-signal for all held CLCs

Low-Level Protocol Layer

The low-level protocol layer exposes direct access to the AGEX Hub API methods. This layer is intended for SDK developers building higher-level abstractions, not for direct use in agent code. It exposes methods corresponding to each Hub API endpoint defined in Section 5.

Configuration

The AGEX SDK is configured via an AgexConfig object or equivalent:

config = AgexConfig(hub_url = "https://hub.agex.api", aid_path = "/path/to/aid.json", privkey_path= "/path/to/privkey.pem", # or hsm_key_id for HSM hsm_enabled = True, log_level = "INFO", audit_endpoint = "https://mycompany.com/agex/audit")

SDK implementations MUST validate the AID signature on startup (not only on first use) to detect AID expiry or corruption early. SDK implementations MUST NOT log credential plaintext under any log level, even DEBUG. SDK implementations MUST store credential plaintext in process memory only (not in files, environment variables, or shared memory).

15f. Advanced Delegation Patterns

The basic delegation model (Section 4.5) covers the common case of a single level of sub-credential creation. This section describes advanced delegation patterns that arise in complex multi-agent systems.

Fan-Out Delegation

In fan-out delegation, a single parent CLC is used to create multiple concurrent sub-CLCs for different child agents. The parent may be an orchestrator managing a pool of worker agents, each needing different sub-scopes of the orchestrator's permissions. The max_concurrent_sub_clcs field in the parent CLC's delegation_policy limits the total number of simultaneously active sub-CLCs, preventing unbounded fan-out that could overwhelm the target service.

Fan-out delegation creates a star topology in the delegation graph: the parent is the centre, with multiple children at depth 1. ERS revocation of the parent cascades to all children simultaneously. This is a desirable property for controlled shutdown of agent pools: revoking the orchestrator's CLC terminates all worker agents' access in a single operation.

Deep Chain Delegation

Deep chain delegation occurs when an agent at depth d delegates to an agent at depth d+1, which in turn delegates to an agent at depth d+2, and so on. AGEX limits chain depth via the max_delegation_depth field at the AID level (hard limit from the Identity Authority) and the max_further_delegation_depth field in each delegation request (soft limit set by the delegating agent).

Deep chain delegation introduces a performance consideration for ERS: the cascade revocation algorithm must traverse the full chain depth. For a chain of depth 5 with fan-out 10 at each level, the cascade may affect up to 10^5 = 100,000 sub-CLCs. Hub implementations MUST handle deep chain traversal efficiently using the chain_provenance index rather than recursive database queries. The recommended implementation uses a transitive closure table maintained as CLCs are issued, allowing the full descendant set to be retrieved in O(1) database lookups.

Cross-Service Delegation

Cross-service delegation arises when an orchestrator holds CLCs for multiple services and wishes to delegate different service credentials to the same child agent. The child agent receives multiple sub-CLCs, one per service, each derived from the parent's CLC for that service. Each sub-CLC has its own clc_id, chain_provenance, and rotation schedule; they are independent credentials that happen to share a common ancestor.

Cross-service delegation is transparent to the protocol: the child agent simply holds multiple CLCs from different services, each obtained through the standard delegation mechanism. The correlation between them (that they were all delegated from the same orchestrator) is visible in the chain_provenance fields and is recorded in the Audit Ledger under the same parent delegation_id.

Scope Reduction Through Chains

A key security property of AGEX delegation is that scope can only decrease through delegation chains, never increase. This property is enforced by: (1) Hub validation that each delegation request's requested_sub_scopes is a subset of the parent's granted_scopes; (2) the sub-CLC's scope_ceiling being set equal to granted_scopes, preventing any future escalation to exceed the parent's scope; and (3) the chain_provenance field in each sub-CLC enabling scope verification by service providers who wish to audit the delegation chain.

A service provider that receives a request authenticated with a sub-CLC can verify the full chain by: retrieving each ancestor CLC via the Hub audit API (or from its own records), verifying that each step in the chain reduced scope, and confirming that the original (root) CLC was issued by the SP to a trust tier-appropriate agent. This chain verification is an optional but recommended security practice for high-risk operations.

15g. Deployment Anti-Patterns

This section documents deployment patterns that violate the AGEX security model or reduce the practical security benefits of the protocol. Implementers SHOULD review their designs against these anti-patterns before production deployment.

Anti-Pattern 1: Over-Permissioned CLCs as a Fallback for Uncertain Scope

Some implementations request a maximal scope set at credential acquisition time to avoid the need for escalation later. This defeats the principle of least privilege that AGEX is designed to enforce. The correct approach is to request only the scopes needed for the current task, use the escalation mechanism if additional scope is needed mid-task, and design agent tasks to be specific enough that their scope requirements are known at the start.

Anti-Pattern 2: Long-Lived CLCs with High rotation_interval_seconds

Setting rotation_interval_seconds to very large values (86400 seconds or more) reduces ARP overhead but extends the window in which a compromised credential can be used before a scheduled rotation provides a natural revocation point. The correct balance is to set rotation intervals based on the risk profile of the credential: payment API credentials SHOULD rotate hourly; read-only analytics credentials MAY rotate daily.

Anti-Pattern 3: Ignoring idle_timeout_seconds

Not setting idle_timeout_seconds in the CLC means credentials remain valid even when the agent is inactive. This violates the AGEX design intent: credentials should be held only while they are in active use. The correct approach is to set idle_timeout_seconds to a value reflecting the maximum expected gap between API calls during normal operation (for example, 300 seconds for a continuously operating agent, 3600 seconds for a batch agent that calls the API periodically).

Anti-Pattern 4: Storing Private Keys in Environment Variables

Environment variables are a common configuration mechanism but are not a secure key store. They are accessible to all processes in the same environment, often logged by monitoring systems, and may be exposed via diagnostic endpoints or error messages. Agent private keys MUST be stored in a hardware security module or an encrypted secrets management system, not in environment variables.

Anti-Pattern 5: Skipping Signature Verification on CLC Receipt

An agent that does not verify provider_signature and hub_signature on received CLCs is vulnerable to a man-in-the-middle attack where the Hub delivers a modified CLC (for example, with narrower scope than approved or a credential that causes attribution confusion). Signature verification is a normative requirement (AC-M-004) and MUST NOT be skipped in production deployments.

Anti-Pattern 6: Single Approval Contact for All Services

Configuring the same email address as the escalation_contact for all services means that approval requests for a payment API escalation arrive in the same inbox as calendar API write requests, making it difficult for approvers to apply appropriate scrutiny to each. The escalation_contact should be a role-specific address: financial operations for payment API escalations, IT security for administrative scope requests, and so on.

Anti-Pattern 7: Delegation Without Monitoring

Creating delegation chains without monitoring the sub-CLCs created means that the scope of the agent system's access can expand significantly without organisational visibility. Hub operators and agent system operators SHOULD monitor the total number of active sub-CLCs under each parent CLC, the scope distribution across the delegation tree, and the depth of delegation chains. Alerts SHOULD fire when these metrics exceed configured thresholds.

Anti-Pattern 8: Using Test Credentials in Production

AGEX supports environment declarations (development, staging, production) in the Intent Manifest. Service providers whose APL policies do not enforce the environment field may inadvertently approve development-tier credentials for production data. APL policies for production services SHOULD include an explicit condition:

```
  { "type": "environment", "operator": "eq", "value": "production" }
```

and route requests with other environment values to rejection or to a separate development-environment credential pool.

15h. Trust Tier Certification Specification

The AGEX Trust Tier System is the mechanism by which AGEX establishes graduated levels of automated trust for agents operating in the AGEX ecosystem. This section specifies the full certification requirements, evaluation criteria, maintenance obligations, and revocation procedures for each trust tier.

Trust Tier 0: Unverified Agent

Tier 0 is the default tier assigned to any agent that has obtained an AGEX AID but has not undergone additional verification. Tier 0 agents may request credentials from any service provider, subject to that provider's APL policy. In practice, well-designed APL policies limit Tier 0 agents to read-only access, non-financial operations, and operations not involving personal data without explicit human approval.

Tier 0 AID issuance requirements: No verification beyond basic AID schema validation and a valid email contact for the owning principal. The IA signs the AID with trust_tier: 0. No documentation of agent purpose, organisational affiliation, or operational history is required.

Tier 0 operational limitations: Hub operators MAY implement rate limiting specifically for Tier 0 agents (for example, 10 credential requests per hour, maximum 1-hour CLC duration) to limit abuse. Service providers whose APL policies auto-approve Tier 1+ requests SHOULD route all Tier 0 requests to human review as a baseline posture.

Trust Tier 1: Organisationally Verified Agent

Tier 1 requires verified organisational identity. The owning organisation must be a legal entity (company, registered non-profit, or government agency) that the issuing IA has verified through one of the following methods: domain-validated organisational certificate (Organisation Validation or Extended Validation TLS certificate from a WebTrust-certified CA), Companies House or equivalent national business registry filing, or direct contractual relationship between the owning organisation and the IA.

Tier 1 AID issuance requirements: IA performs organisational identity verification. AID's principal fields must accurately reflect the verified organisation. The agent's purpose (capabilities, type, and intended services) must be declared and reviewed by the IA, though the IA does not verify the accuracy of capability claims. Contact information for the owning organisation's technical and security teams must be provided and validated (e.g., by verifying the contact can receive email at the declared domain).

Tier 1 operational characteristics: Most service providers' APL policies auto-approve Tier 1 requests for read-only and standard write operations during business hours. Maximum CLC duration for Tier 1 agents SHOULD be 24 hours. Tier 1 agents MAY receive delegation permissions, subject to the parent CLC's delegation_policy.

Trust Tier 2: Audited Agent

Tier 2 requires both organisational verification (as for Tier 1) and a security audit of the agent system. The audit evaluates: the agent's credential handling practices (specifically, correct implementation of AGEX ARP and ERS self-signalling), the agent's data handling practices relative to its declared intent capabilities, the owning organisation's security programme (SOC 2 Type II or ISO 27001 certification is recommended but not required), and demonstrated operational history (minimum 90 days of operating at Tier 1 without security incidents).

The audit is performed by an AGEX Foundation-approved auditor or by the IA's own audit team (for IAs that have this capability certified by the Foundation). Audit scope includes review of agent source code or binaries (where auditor has access), review of credential storage implementation, review of key management procedures, and review of anomaly detection and alerting configurations.

Tier 2 AID issuance requirements: Written audit report from an approved auditor, signed by a qualified security professional. The report must contain a specific statement that the agent implementation correctly implements AGEX credential lifecycle management. The IA reviews the audit report and makes an independent determination of Tier 2 eligibility.

Tier 2 maintenance: Tier 2 status must be renewed annually. The renewal requires a re-audit or a self-attestation with evidence (for agents with no security incidents in the previous year). Material changes to the agent's credential handling implementation MUST trigger a re-audit notification to the IA; the IA MAY downgrade to Tier 1 pending re-audit.

Trust Tier 3: Mission-Critical Certified Agent

Tier 3 is the highest trust tier and is reserved for agents operating in regulated industries or critical infrastructure contexts. Tier 3 requires all Tier 2 requirements plus: demonstrated compliance with at least one relevant regulatory framework (PCI DSS Level 1, HIPAA Business Associate Agreement, GDPR Article 30 records, FCA or SEC registration for financial services agents), formal penetration testing of the agent system by an independent third party within the past 12 months with no unresolved critical findings, and executive sponsorship from the owning organisation (C-suite or VP-level officer attests to the accuracy of the agent's AID declarations and accepts personal accountability for security incidents).

Tier 3 AID issuance requirements: All Tier 2 requirements, plus: penetration test report, regulatory compliance certification, executive attestation letter. The AGEX Foundation Technical Steering Committee reviews all Tier 3 applications.

Tier 3 operational characteristics: Tier 3 agents receive the highest level of automated trust from APL policies. Many service providers will auto-approve Tier 3 agents for all scopes below the admin tier without human review. Tier 3 agents may receive longer CLC durations (up to 7 days for appropriate use cases) and higher rate limits.

Trust Tier Revocation

An IA may revoke or downgrade a trust tier at any time. Reasons for tier downgrade or revocation include: security incident disclosure, audit findings identifying implementation defects, failure to renew annual certification, regulatory action against the owning organisation, or credible evidence of the agent being used in violation of its declared intent. Tier downgrade is implemented by publishing a revised AID (with lower trust_tier) to the Hub. Outstanding CLCs issued to the old AID are not automatically revoked by a tier downgrade; they remain valid until their next rotation cycle, at which point the new lower tier applies to the rotation request evaluation.

15i. AGEX Foundation Certification Programme

The AGEX Foundation Certification Programme establishes and enforces quality and security standards for three categories of AGEX participants: Identity Authorities (IAs), Hub Operators, and Service Providers.

Identity Authority Certification

An organisation wishing to operate as an AGEX Identity Authority must: be a legal entity in good standing in its jurisdiction of incorporation, have a demonstrated information security programme (SOC 2 Type II or ISO 27001 certification required), operate a physically secure key ceremony process for the IA signing key (witnessed by an AGEX Foundation representative or approved auditor), agree to the AGEX IA Terms of Service (which include mandatory AID revocation obligations, audit cooperation, and breach notification requirements), and demonstrate the technical capability to correctly issue and revoke AIDs by passing the AGEX IA certification test suite.

IAs are certified for a 2-year term, renewable upon passing the annual renewal audit. The Foundation publishes the list of certified IAs in its Root of Trust registry, accessible at root.agex.api. The IA's public key is included in the registry entry; Hubs use this registry to verify AID signatures from IAs they have not previously interacted with.

IAs operate under strict liability for incorrect trust tier assignments. An IA that certifies a Tier 2 or Tier 3 agent that subsequently demonstrates the inability to correctly implement AGEX credential lifecycle management is liable to the AGEX Foundation for the costs of security incident response attributable to the incorrect certification. This liability creates a strong incentive for rigorous audit processes.

Hub Operator Certification

An organisation wishing to operate an AGEX Hub must: satisfy the same legal entity and security programme requirements as IAs, demonstrate technical compliance with all normative Hub requirements (HC-M-001 through HC-M-013 and recommended HC-O-001 through HC-O-005) by passing the AGEX Hub certification test suite, operate Hub infrastructure in a data centre with physical access controls and 24x7 monitoring, maintain a published Security Incident Response policy with maximum 24-hour response time for confirmed credential security incidents, and agree to the AGEX Hub Operator Terms of Service (which include data residency obligations, SLA commitments, and cooperation with law enforcement under applicable legal process).

Hub Operators are certified at two tiers: Community Hub (suitable for development and small-scale deployments, lower infrastructure requirements) and Enterprise Hub (suitable for production deployments serving organisations with regulatory compliance requirements, requires geographically redundant infrastructure and 99.99% uptime SLA).

Service Provider Certification

Service Provider certification is lighter-weight than IA or Hub certification, reflecting the fact that the SP is the issuing authority for credentials and bears primary responsibility for its own credential security. SP certification requires: correct implementation of SC-M-001 through SC-M-008 verified by the Hub operator during onboarding, publication of an APL policy at the registered policy_endpoint, and agreement to the AGEX SP Terms of Service.

The AGEX Foundation maintains a public registry of certified SPs, searchable by service category, geographic jurisdiction, and minimum supported trust tier. This registry serves as the basis for the Hub's service provider discovery endpoint.

15j. Security Incident Response Procedures

This section specifies the procedures for handling AGEX-related security incidents. Incidents are classified by severity and response procedures differ accordingly.

Severity Classification

Severity 1 (Critical): Active credential exfiltration or abuse by an adversary, confirmed compromise of a Hub operator's signing key, compromise of an IA's signing key, or compromise that enables forgery of CLCs or AIDs. Response: immediate containment via ERS, notification to affected parties within 1 hour, AGEX Foundation notification within 2 hours, public disclosure within 72 hours (or as required by applicable regulation).

Severity 2 (High): Suspected but unconfirmed credential compromise, single agent compromise affecting fewer than 100 CLCs, APL policy misconfiguration resulting in over-permissioned CLCs, or ARP rotation failures affecting more than 10% of an agent population. Response: ERS invocation for affected agents within 4 hours, root cause analysis initiated within 24 hours, affected parties notified within 48 hours.

Severity 3 (Medium): Audit Ledger integrity anomaly, rate limit abuse without evidence of credential compromise, or delegation chain anomalies. Response: investigation initiated within 24 hours, Hub operator notification within 48 hours.

Severity 4 (Low): Single rotation failure with no impact on credential validity, performance degradation affecting non-ERS subsystems, or false positive anomaly detection alerts. Response: tracked in issue management system, addressed in next release cycle.

Incident Response Roles and Responsibilities

The AGEX security incident response involves three organisational roles. The Incident Coordinator is the person responsible for overall incident management; in Hub operator incidents this is the Hub operator's security team; in IA incidents this is the IA's security team. The Technical Lead is responsible for the technical containment and remediation actions. The AGEX Foundation Liaison is responsible for communicating with the AGEX Foundation, coordinating multi-party incidents, and managing public disclosure.

For Severity 1 incidents, all three roles must be staffed immediately and a war room (virtual or physical) established. An incident status page SHOULD be published at status.agex.api (operated by the Foundation) within 2 hours of Severity 1 classification.

Post-Incident Analysis

All Severity 1 and Severity 2 incidents require a post-incident analysis report submitted to the AGEX Foundation within 30 days of resolution. The report MUST cover: timeline of events from initial detection to full resolution, root cause analysis, scope of impact (number of agents and CLCs affected, services involved), evidence of remediation completeness, and lessons learned with protocol or implementation changes proposed. The Foundation publishes anonymised summaries of post-incident analysis reports in its annual security report, contributing to community-wide knowledge about AGEX deployment risks.

15k. AGEX Policy Language: Advanced Features

This section extends the APL specification in Section 6 with advanced policy features that address complex operational requirements encountered in enterprise and regulated industry deployments.

Policy Inheritance and Composition

Service providers with multiple API tiers often need to express layered policies where a base policy applies to all request types, with specialised policies overlaid for specific scope categories or agent types. APL supports this through policy composition via the extends field.

A policy document with an extends field references the policy_id of a base policy document. During evaluation, the extending policy's rules are evaluated first (in priority order); if no rule in the extending policy matches, evaluation falls through to the base policy's rules. This allows service providers to maintain a single base policy and create lightweight override policies for specific deployments (for example, a policy for a specific enterprise customer's agents that grants higher trust than the default policy for the same tier).

The extends chain may be at most three levels deep to prevent circular references and to bound evaluation time. The Hub MUST detect and reject circular extends chains at policy registration time.

Dynamic Conditions Based on Agent History

APL 1.0 does not support dynamic conditions that reference an agent's historical credential usage (for example, "approve if this agent has successfully operated under a CLC for this service for more than 30 days"). This limitation is intentional: dynamic conditions require the policy engine to query external state, breaking the determinism requirement. However, the Hub's trust tier system provides a mechanism for incorporating historical behaviour: IAs may consider operational history when assigning trust tiers, allowing historical context to flow into the static AID trust_tier field and be evaluated by APL conditions.

A future APL version will consider a read-only history query API that allows policy conditions to reference pre-computed agent behavioural metrics (such as mean_rotation_success_rate or days_since_last_security_incident) calculated by the Hub and injected into the evaluation context. This is tracked as Open Issue 6 for Working Group deliberation.

Policy Versioning and Change Management

Service providers operating production systems need to update their APL policies without disrupting existing CLCs. APL policy changes SHOULD be managed through the following versioning procedure.

When a policy update is proposed, the new policy document is uploaded to the Hub's policy staging endpoint as a candidate policy. The Hub assigns the candidate policy a policy_candidate_id and makes it available for testing via the Hub's policy testing endpoint (where SPs can test the candidate against synthetic requests without affecting production). After the SP has verified the candidate policy against its test corpus, the SP promotes the candidate to active status via the Hub's policy promotion endpoint.

Promotion is atomic: the Hub switches from the old active policy to the new active policy for all subsequent credential requests. Outstanding CLCs issued under the old policy are not retroactively affected by the policy change; their terms are fixed at issuance time (recorded in the manifest_hash in each CLC).

The Hub records each policy promotion event in the Audit Ledger with: the old policy_id, the new policy_id, the SHA-3-256 hashes of both policy documents, and the timestamp. This record enables auditors to verify which policy governed each CLC issuance by comparing CLC issuance timestamps against policy promotion history.

Time-Based Policy Scheduling

Some service providers need policies to change on a schedule. For example, a financial services provider may allow Tier 1 agents during business hours but require human review at night and weekends. Rather than requiring the SP to push policy updates on a schedule, APL's time_window condition enables this directly within a single policy.

A complementary feature, policy_schedule, allows SPs to register multiple candidate policies with specific activation timestamps. The Hub automatically promotes the scheduled policy at the registered time. This is useful for planned policy changes (for example, tightening policy during a scheduled maintenance window) without requiring operational intervention at activation time.

Policy Debugging and Audit

SPs may request a policy evaluation trace from the Hub for any recent credential request. The trace shows: which rules were evaluated (in priority order), the boolean result of each condition in each evaluated rule, the final decision, and the elapsed evaluation time. Traces are available for 7 days after the evaluated request. This facility supports debugging of policy misconfigurations without requiring policy changes to add logging.

The policy evaluation trace is subject to the same access controls as the audit log: the SP may access traces for requests targeting its services; the requesting agent's owning organisation may access traces for its own agents' requests; Hub operators may access all traces.

15l. AGEX in Regulated Industries

This section provides a detailed analysis of AGEX deployment in five regulated industry contexts: financial services, healthcare, critical infrastructure, government/public sector, and legal services. Each analysis covers the specific regulatory requirements that AGEX addresses and deployment recommendations.

Financial Services

Financial services AI agents operate under regulations including the EU MiFID II (Markets in Financial Instruments Directive), the UK FCA's AI and Machine Learning guidance, the US SEC's AI use in investment advice regulations, and PCI DSS for payment card handling.

AGEX addresses financial services regulatory requirements in several ways. The Intent Manifest's task_type and data_classification fields provide a standardised vocabulary for declaring the nature of financial data operations, which can be directly mapped to MiFID II's concept of investment service categories. The escalation thresholds for transaction_value_usd enable implementation of the human oversight requirements in AI-assisted trading and advisory contexts: any agent action above a configurable monetary threshold is automatically referred for human review before the credential is issued to complete the transaction.

For PCI DSS compliance, AGEX's trust tier system provides a framework for satisfying PCI DSS Requirement 7 [OWASP-API] (restrict access to system components and cardholder data by business need to know) and Requirement 8 (identify users and authenticate access to system components). AGEX CLCs provide the credential management layer required by PCI DSS Requirement 8.3, and the ARP rotation mechanism satisfies the requirement for credential rotation without the operational burden of manual rotation management.

Deployment recommendation: Financial services agents SHOULD operate at Tier 2 minimum for any operation involving customer account data, and Tier 3 for payment processing. APL policies for payment APIs

MUST include scope ceiling restrictions that prevent Tier 0 or Tier 1 agents from obtaining payment:write or refund:write scopes.

Healthcare

Healthcare AI agents operate under regulations including HIPAA (Health Insurance Portability and Accountability Act) in the US, the EU's GDPR applied to health data (special category data under Article 9), and the UK's Data Security and Protection Toolkit requirements for NHS suppliers.

AGEX's data_handling fields in the Intent Manifest provide direct support for HIPAA's Minimum Necessary standard (45 CFR 164.502(b)): an agent requesting access to patient records MUST declare the minimum scope of PHI (Protected Health Information) access needed for the specific task. Service providers can enforce this standard by checking the declared intent against the requested scopes in the APL policy.

The Audit Ledger provides a machine-queryable record that satisfies HIPAA's Access Audit Controls requirement (45 CFR 164.312(b)), which requires healthcare organisations to implement hardware, software, and procedural mechanisms to record and examine access to ePHI (electronic Protected Health Information). Each CLC access event is recorded with the agent's AID, the accessed service, the granted scopes, and the timestamp.

Deployment recommendation: Any agent processing PHI MUST declare pii_processing: true and pii_categories: ["health_data"] in the manifest. APL policies for healthcare APIs MUST require trust_tier >= 2 for any PHI access scope, and MUST require data_handling.retention_policy to be "no_retention" or "session" for agents without explicit long-term data retention authorisation.

Critical Infrastructure

Critical infrastructure AI agents (energy grid management, water treatment, transportation systems) operate under the most stringent security requirements of any deployment context. In the EU, the NIS2 Directive (Directive 2022/2555) [EU-AIA] imposes specific requirements on operators of essential services. In the US, CISA's Critical Infrastructure Protection framework applies.

AGEX's ERS mechanism is particularly valuable in critical infrastructure contexts because it provides a rapid, coordinated response to suspected agent compromise. A compromised agent controlling industrial control systems could cause physical harm; the ability to revoke all credentials within 60 seconds across all connected systems is a material safety control.

The delegation chain mechanism is important for critical infrastructure because it enables the implementation of a hierarchical control structure (a supervisory agent delegates to control agents, which delegate to sensor/actuator agents) with cryptographic scope enforcement, ensuring that a compromised actuator agent cannot exceed its authorised scope even if the compromise is not immediately detected.

Deployment recommendation: Critical infrastructure agents MUST operate at Tier 3. CLCs for control systems MUST have rotation_interval_seconds of no more than 3600 seconds (1 hour). APL policies MUST deny all write scopes to agents that have not declared human_visible: true in their manifest, ensuring that machine-to-machine control actions are always declaratively subject to human visibility.

Government and Public Sector

Government AI agents operate under government-specific security frameworks including NIST 800-53 [NIST-AI] (Security and Privacy Controls for Information Systems and Organisations) in the US, the UK Cyber Essentials scheme, and national-level classification systems for sovereign data.

AGEX's geographic restriction capabilities (aid.restrictions.geo_restrictions, manifest data_handling.transfer_jurisdictions, and APL policy geographic conditions) provide a technical mechanism for enforcing data sovereignty requirements. A government APL policy can require that agents serving sensitive data are deployed within the national jurisdiction, preventing cross-border data flows that would violate sovereignty rules.

The trust tier certification system, specifically the government's ability to operate its own IA for government agents, enables a sovereign identity infrastructure where government AI agents' identities are certified entirely within the government's own systems without dependency on commercial Identity Authorities.

Legal Services

Legal AI agents processing privileged communications (attorney-client privilege, legal professional privilege) operate under legal professional conduct rules that impose confidentiality obligations. AGEX's data_classification: "restricted" designation, combined with APL policies that restrict access to privilege-reviewed agents, provides a technical layer of enforcement for these obligations.

The Audit Ledger is particularly important in legal contexts because it creates a verifiable record of which agents accessed what information and when, which may be relevant in privilege disputes or in demonstrating that confidentiality obligations were met. Legal services organisations SHOULD implement external anchoring of the Audit Ledger hash chain to provide independent verification that the audit record has not been tampered with after the fact.

15m. AI Agent Framework Integration Patterns

This section describes integration patterns for the four most widely deployed AI agent frameworks as of early 2026: LangChain, AutoGen, CrewAI, and n8n. Each integration pattern is described at the API level with representative code structure. These integrations are provided as non-normative guidance to accelerate adoption.

LangChain Integration

LangChain implements AI agents as sequences of tool calls within a LangChain Agent Executor. AGEX integration is implemented as a LangChain Tool that wraps API calls with automatic AGEX credential management. The integration intercepts each tool's HTTP call, checks whether an active CLC exists for the target service, acquires one if not (via agex.acquire()), injects the credential into the HTTP headers, and triggers ARP rotation if the current CLC is within its rotation window.

The recommended integration pattern uses LangChain's callback system to hook into the pre-tool and post-tool events. The pre-tool hook acquires the credential; the post-tool hook records the API call against the CLC's usage audit. On agent completion, the post-run hook calls agex.release() for all CLCs acquired during the run.

A LangChain AGEX CredentialStore class maintains the mapping between tool names and service IDs, and between service IDs and active CLCs. The CredentialStore is instantiated once per agent run and is not shared between parallel agent instances.

AutoGen Integration

AutoGen implements multi-agent conversations where agents communicate via message passing. AGEX integration for AutoGen is implemented at the ConversableAgent level: each AutoGen agent that makes API calls is configured with an AgexCredentialManager that handles credential acquisition for the agent's registered service IDs.

The AutoGen integration uses AGEX's delegation mechanism to implement AutoGen's hierarchical agent patterns: when a UserProxyAgent initiates a task, it acquires a parent CLC; when it delegates to an AssistantAgent for a sub-task, it creates a sub-CLC via agex.delegate() and passes it to the child agent via the AutoGen message channel. The child agent's AgexCredentialManager recognises the sub-CLC and uses it without making a separate credential request.

This implementation correctly models AutoGen's authority hierarchy in the AGEX protocol: the UserProxyAgent holds the root credential; child agents hold derived credentials with constrained scope. ERS revocation of the UserProxyAgent's CLC cascades to all child agents.

CrewAI Integration

CrewAI implements agent systems as "crews" of agents with defined roles, tasks, and tools. AGEX integration in CrewAI is implemented at the Tool level: CrewAI tools that call external APIs are wrapped with an AgexTool decorator that handles credential management.

The AgexTool decorator inspects the tool's service_id annotation, checks the AGEX credential store for a valid CLC, acquires one if needed, and injects the credential into the API call. For CrewAI crews with a manager agent, the manager agent holds parent CLCs for all services used by the crew; worker agents receive delegated sub-CLCs via the AGEX delegation mechanism.

CrewAI's Process.hierarchical mode aligns naturally with AGEX's delegation model: the manager agent at the top of the hierarchy holds the widest credentials; worker agents receive scoped sub-credentials matched to their assigned role's permissions.

n8n Integration

n8n is a workflow automation platform that can execute AI agent workflows as graphs of connected nodes. AGEX integration in n8n is implemented as a dedicated AGEX node set: the AGEX Credential Request node, the AGEX Rotation node, and the AGEX Revoke node. These nodes can be placed in n8n workflows to manage the credential lifecycle around API call nodes.

The AGEX Credential Request node takes a service_id, intent parameters, and AID path as inputs, executes the AGEX credential acquisition flow, and outputs the credential value and clc_id for use in subsequent nodes. The Rotation node can be triggered by an n8n schedule and executes ARP rotation for a specified clc_id. The Revoke node initiates ERS self-revocation and is typically placed as the final node in a workflow.

For n8n users building agent workflows, the recommended pattern is: place an AGEX Credential Request node at the start of the workflow, connect its credential output to the API call nodes via n8n expressions, and place an AGEX Revoke node in the error and success branches to ensure credentials are released regardless of workflow outcome.

15n. Protocol Message Timing Analysis

This section provides a detailed timing analysis of all AGEX protocol message exchanges, establishing the latency budget for each flow and identifying the critical path components that determine end-to-end latency.

Credential Acquisition: Timing Budget

The credential acquisition flow involves five network hops in the auto-approve case: Agent to Hub (AH), Hub AID validation (internal), Hub APL policy evaluation (internal), Hub to SP (HS), and SP to Hub (SH), and Hub to Agent (HA). The round-trip time from the agent's perspective is approximately: RTT(AH+HA) + RTT(HS+SH) + internal.

For a Hub and SP co-located in the same data centre region, the internal processing time typically dominates. Target latency decomposition:

Network: Agent-to-Hub round trip: 20–50ms (same region), 80–200ms (cross-region) AID validation (cache hit): 2ms; (cache miss, revocation check): 15ms APL evaluation (< 50 rules): 5ms; (50-100 rules): 25ms; (> 100 rules): 50ms Hub-to-SP forwarding: 5–30ms (same region); 50–150ms (cross-region) SP CLC generation (envelope encryption): 10ms; (key ceremony via HSM): 50ms SP-to-Hub response: 5–30ms (same region) Hub-to-Agent final delivery: included in initial round-trip

Total auto-approve, same-region, AID cached, small policy: ~50ms Total auto-approve, cross-region, AID cache miss, large policy, HSM: ~450ms Total pending_human_approval response: ~200ms (to return pending status)

The 99th percentile latency target of 500ms for SP-round-trip auto-approve requests is achievable within the above budget if Hub and SP are deployed in the same cloud region. Hub operators SHOULD recommend that service providers deploy their AGEX endpoints in the same region as the Hub they register with.

ARP Rotation: Timing Budget

The ARP rotation flow adds one additional network round-trip (Hub-to-SP and SP-to-Hub for the rotation_forwarding and rotation_response messages) to the agent-Hub round-trip. The atomic swap at the agent adds local computation time for new keypair generation (~2ms with CSPRNG) and ECDH-ES decryption of the new envelope (~5ms).

Total ARP rotation, same-region: ~150ms Total ARP rotation, cross-region: ~600ms

The 2,000ms 99th percentile target provides comfortable margin for cross-region deployments and for SP-side processing latency under load.

ERS: Timing Budget

The ERS 60-second completion bound is dominated by the SP notification phase. For a hub managing CLCs across 50 service providers (a typical enterprise deployment), the notification phase requires 50 parallel HTTP calls. The critical path is the slowest SP to acknowledge.

For 50 SPs in the same region: expected completion 2-5 seconds; 99th percentile 15 seconds. For 50 SPs across multiple regions: expected completion 5-15 seconds; 99th percentile 45 seconds. For 200 SPs: expected completion 10-30 seconds; 99th percentile approaching 60 seconds.

Hub operators managing more than 100 service providers per tenant SHOULD implement SP notification concurrency controls that prioritise SP acknowledgment speed (preferring SPs with recent fast responses) over alphabetical or registration-order notification.

Clock Skew Impact on Timing

The 300-second timestamp tolerance is the primary mechanism preventing replay attacks. Its effect on legitimate traffic is minimal if NTP synchronisation maintains clocks within 60 seconds of consensus time. The 300-second tolerance provides a 5x safety margin for NTP synchronisation failures of up to 60 seconds. Agents experiencing repeated TIMESTAMP_OUT_OF_RANGE errors SHOULD log the Hub's timestamp from error responses and compare against their own clock to diagnose synchronisation issues.

15o. Comparison with Emerging Machine Identity Standards

This section positions AGEX relative to emerging machine identity standards and industry initiatives that address overlapping aspects of the autonomous system credential management problem.

Sigstore / OIDC-Based Machine Identity

Sigstore (sigstore.dev) provides a signing infrastructure for software artefacts using OIDC-based identity. In Sigstore's Cosign tool, a software signing event is tied to an OIDC identity (such as a GitHub Actions workflow identity or a human developer identity), providing a transparency log record of who signed what and when.

Sigstore addresses supply chain security for software artefacts, not runtime credential management for executing agents. The overlap with AGEX is at the identity layer: AGEX AIDs and Sigstore OIDCidentities are both mechanisms for establishing verifiable identity for automated processes. AGEX is specifically designed for runtime credential management once a deployed agent is executing; Sigstore is designed for pre-deployment artefact signing. The two can be complementary: a Sigstore-signed container image establishes the agent's software supply chain integrity; an AGEX AID establishes the running container's runtime credential identity.

CAEP (Continuous Access Evaluation Protocol) [RFC9396]

The OpenID Foundation's CAEP (Continuous Access Evaluation Protocol) [RFC9396] and IETF's SSE (Shared Signals and Events) framework define mechanisms for communicating security events between identity providers and relying parties, enabling real-time revocation of access tokens when security conditions change.

CAEP is conceptually aligned with AGEX's ERS: both address the problem of revoking access rapidly when security conditions change. The key differences are scope and architecture. CAEP operates within the OAuth 2.0 / OIDC ecosystem, communicating security events between authorisation servers and resource servers for human-authenticated sessions. AGEX ERS operates at the level of agent credential relationships, communicating revocation events from the Hub to all service providers holding CLCs for the affected agent, with the cascade mechanism extending to delegation chains.

Future interoperability between AGEX ERS and CAEP receivers is feasible: a Hub could emit CAEP-formatted security events alongside its proprietary ERS signals, enabling AGEX-integrated service providers and non-AGEX OAuth 2.0 service providers to receive the same revocation information through their respective native channels.

WIMSE (Workload Identity in Multi-System Environments)

The IETF WIMSE working group (Workload Identity in Multi-System Environments) is developing standards for workload identity that address cross-domain scenarios beyond SPIFFE's scope. WIMSE's problem statement has significant overlap with AGEX: both address identity and credential management for automated software processes operating across organisational boundaries.

The key distinction is that WIMSE focuses on identity propagation (how a workload's identity is carried across service calls), while AGEX focuses on credential lifecycle management (how credentials are acquired, rotated, delegated, and revoked). WIMSE and AGEX are complementary: WIMSE could provide the transport-layer identity propagation mechanism for AGEX agents when making API calls, while AGEX provides the credential lifecycle management layer above WIMSE.

The AGEX Foundation intends to engage with the WIMSE working group to explore alignment between the two approaches, with the goal of ensuring that AGEX credentials can be carried using WIMSE's identity propagation mechanisms where appropriate.

15p. AGEX Ecosystem Economic Model

This section analyses the economic structure of the AGEX ecosystem, examining the incentive mechanisms that drive adoption, the cost structure of deployment, and the value distribution between participants.

Value Creation and Distribution

AGEX creates value for four categories of participant. For agent-owning organisations, AGEX eliminates the engineering cost of custom credential management infrastructure (typically 2-6 months of senior engineer time for a robust custom solution), reduces security incident costs by eliminating the most common credential management failure modes, and reduces compliance overhead by providing a structured audit trail that satisfies multiple regulatory requirements simultaneously.

For service providers, AGEX reduces the cost of onboarding API consumers in agentic deployments (replacing manual credential management with a standardised automated protocol), reduces security incident costs by enabling rapid credential revocation, and provides a competitive differentiation signal for API providers targeting AI-native customers.

For Hub operators, AGEX creates a viable business model as a managed infrastructure provider. Hub operators charge for hosting the Hub infrastructure, providing SLA guarantees, operating the AID registry and revocation infrastructure, and providing managed compliance features (audit log retention, regulatory reporting). The Community Hub tier enables lower-cost access for smaller deployments; the Enterprise Hub tier commands premium pricing for the compliance and reliability features required by regulated industry customers.

For Identity Authorities, AGEX certification services create a new revenue stream for existing PKI operators, trust service providers, and security auditing firms. The Tier 2 and Tier 3 certification process involves significant audit work that commands professional services fees comparable to SOC 2 Type II audit engagements.

Cost Structure

The primary cost components for a production AGEX deployment are: Hub infrastructure (compute, storage, network for the Hub service itself), IA certification (one-time and annual renewal), developer integration cost (estimated 2-4 weeks for SDK integration into an existing agent framework), and operational monitoring.

At scale, the Hub infrastructure cost per CLC lifecycle is modest. A Hub processing 1 million CLC issuances per day with an average 24-hour CLC lifetime (1 million active CLCs at steady state) requires approximately 4 vCPU and 16GB RAM for the API and processing tiers, plus storage for the audit log (estimated 500 bytes per event, with approximately 5 events per CLC lifecycle = 2.5GB per day at this scale). Cloud infrastructure costs at this scale are approximately $500-1000 per month.

Open Source and Commercial Hub Implementations

The AGEX Foundation publishes an open source reference Hub implementation under the Apache 2.0 licence. This implementation is suitable for development, testing, and small-scale production deployments.

Commercial Hub operators are expected to build on the reference implementation with added reliability, compliance, and operations features, operating as a managed service.

This dual-track model (open source reference + commercial managed service) follows the successful pattern established by Kubernetes (open source core + commercial managed Kubernetes services), Kafka (open source core + commercial managed streaming), and HashiCorp Vault (open source core + enterprise). The open source core ensures that AGEX does not become dependent on any single commercial provider, while commercial managed services lower the operational burden for enterprises that prefer not to operate their own infrastructure.

15q. Schema Evolution and Migration

Protocol schemas evolve over time as new use cases are discovered and as operational experience reveals gaps in the initial design. This section describes the principles and procedures governing AGEX schema evolution.

Schema Versioning Principles

AGEX schema versions are managed independently from the protocol version. The AID schema, Intent Manifest schema, CLC schema, ARP message schemas, and delegation schemas each carry their own version field (e.g., aid_version: "1.0"). This allows individual schema components to evolve independently without forcing a full protocol version bump.

The following rules govern schema evolution. Fields MAY be added to any schema in a minor version update, provided they are optional (not required for correct protocol operation). Fields MUST NOT be removed or renamed within a major version. The semantic meaning of existing fields MUST NOT change within a major version. New enum values MAY be added to existing enum fields within a minor version; implementations MUST treat unknown enum values as ignorable (not as fatal errors) to support forward compatibility.

Field Addition Process

New fields proposed for AGEX schemas are submitted to the Technical Steering Committee as an Enhancement Proposal. The proposal must include: the field name and type, the motivation (what use case the field enables), the default value for implementations that do not set the field, the impact on existing implementations, and at least one reference implementation demonstrating the field's use.

The TSC reviews proposals quarterly. Accepted proposals are assigned to the next minor version release. The review period from proposal submission to inclusion in a released version is typically 6-9 months, providing implementers with adequate notice.

Backwards Compatibility Guarantees

AGEX 1.x is fully backwards compatible within the 1.x series. An AGEX 1.0 agent can successfully request credentials from an AGEX 1.1 Hub; the Hub accepts the 1.0 request and processes it using 1.0 semantics for missing 1.1 fields. An AGEX 1.1 agent can successfully request credentials from an AGEX 1.0 Hub; the Hub ignores 1.1-specific fields it does not recognise (as required by the forward compatibility rules), returning a 1.0 CLC. The agent detects the 1.0 CLC and operates in compatibility mode.

Backwards compatibility is tested by the conformance test suite, which includes a cross-version compatibility matrix: each supported version is tested against every other supported version, verifying that cross-version interactions produce correct outcomes for the lower-version participant.

Deprecation Policy

Fields deprecated in a minor version are documented as deprecated but retained until the next major version. The deprecation notice in the specification includes: the version in which the field was deprecated, the reason for deprecation, and the recommended alternative. Hub implementations MUST continue to accept deprecated fields until they are removed in the next major version.

The intent is that major version transitions are rare (no more than once every 5 years) and are announced with a minimum 18-month migration timeline. Within a major version, implementers can depend on schema stability.

15r. Reference Implementation Architecture

The AGEX Foundation publishes an open source reference Hub implementation designed to demonstrate correct protocol implementation and to serve as the basis for conformance testing. This section describes the architecture of the reference implementation.

Technology Stack

The reference Hub implementation uses the following technology stack, selected for broad availability and community familiarity rather than maximum performance: Go 1.22 (the server implementation language), PostgreSQL 15 (the primary data store for CLCs, AIDs, and Audit Ledger), Redis 7 (the in-memory cache for AID validation and the ERS CLC graph), and NATS JetStream (the message bus for the Rotation Coordinator and ERS subsystems).

The API layer is implemented as a standard Go HTTP server using the net/http package without additional web frameworks, to minimise dependency surface area and demonstrate that AGEX does not require specific web framework features. The cryptographic operations use Go's standard library crypto/ed25519 and golang.org/x/crypto/chacha20poly1305 packages.

Repository Structure

The reference implementation repository (github.com/agex-foundation/agex-hub) is organised as follows:

cmd/agex-hub/ Main binary entrypoint and configuration loading internal/api/ HTTP API handlers (one file per endpoint group) internal/identity/ AID validation, IA registry, revocation index internal/broker/ Credential request routing and CLC delivery internal/rotation/ ARP state machine and rotation scheduling internal/policy/ APL policy engine and policy store internal/audit/ Audit Ledger write and query implementation internal/ers/ Emergency Revocation System internal/federation/ AHFP cross-Hub protocol internal/crypto/ Cryptographic utility functions (not primitives) pkg/agexschema/ Public Go package for AGEX schema types pkg/agexclient/ Reference agent client library tests/conformance/ Automated conformance test suite tests/integration/ End-to-end integration tests docs/ Protocol specification (this document) and API docs

Key Design Decisions in the Reference Implementation

The ERS subsystem is implemented as a separate Go binary (cmd/agex-ers) that communicates with the main Hub via the NATS message bus. This physical separation ensures that ERS process failures do not affect normal credential operations, and that ERS can be independently scaled and monitored.

The Audit Ledger is implemented using PostgreSQL's append-only write mode with row-level security policies that prevent UPDATE and DELETE operations on audit records. The hash chain is maintained in a Go background goroutine that reads unprocessed events from a staging table, computes the chain hash, and writes the completed event to the immutable audit table. This design allows high write throughput (staging table accepts concurrent writes) while maintaining the sequential chain property (the background goroutine is the sole writer to the audit table).

The Policy Engine is implemented as a pure function: func Evaluate(policy Policy, aid AID, manifest Manifest) Decision. The pure function design enables straightforward testing (the policy test harness simply calls Evaluate with synthetic inputs), parallelisation (multiple policy evaluations can run concurrently with no shared state), and caching (the same policy, AID hash, and manifest hash tuple always produces the same decision).

Testing and Conformance

The conformance test suite (tests/conformance/) is the primary tool for verifying third-party Hub implementations against this specification. The test suite is designed to be Hub-implementation-agnostic: it interacts with the Hub under test only through the AGEX HTTP API, making no assumptions about the internal implementation. The suite covers all mandatory conformance requirements (HC-M-001 through HC-M-013) plus the optional requirements (HC-O-001 through HC-O-005).

Third-party Hub implementations are invited to submit their conformance test results to the AGEX Foundation for certification. The Foundation verifies submissions by running the conformance suite against the submitting implementation in a controlled environment and issues a certification certificate upon successful completion. Certified implementations are listed in the Foundation's Hub Registry.

15s. Detailed Future Roadmap

This section expands on the governance section's implementation roadmap with detailed technical specification of the features planned for future AGEX versions.

AGEX 1.1 Planned Features

Three features are targeted for AGEX 1.1, planned for Q4 2026: automated AID renewal, the AGEX Credential Freshness Token (CFT) mechanism, and APL policy inheritance.

Automated AID Renewal: An AID renewal protocol analogous to ACME (Automatic Certificate Management Environment, RFC 8555) for X.509 certificates. An agent approaching its AID expiry would automatically request a new AID from the issuing IA, presenting its current valid AID as proof of existing identity. The IA would verify the renewal eligibility (no tier change required, no adverse history), generate a new AID with a fresh aid_id and updated expires_at, and publish it to the Hub. The renewal protocol would include a proof-of-possession step ensuring that the renewing agent holds the private key corresponding to the current AID.

AGEX Credential Freshness Token (CFT): A lightweight mechanism for obtaining a fresh credential value without the full ARP rotation overhead. The CFT mechanism would allow an agent holding a valid CLC to request a short-lived credential (valid for 5-15 minutes) from the Hub, encrypted under the agent's current public key. The CFT would not update the CLC's rotation counter or trigger the ARP state machine; it would be a stateless read from the SP's credential generation system. CFTs address the use case of agents making infrequent API calls who want fresh credentials for each call without the overhead of full ARP rotation.

APL Policy Inheritance: The policy composition feature described in the APL Advanced Features section (Section 15k), enabling SPs to define base policies and override policies without duplicating rule sets.

AGEX 2.0 Considerations

AGEX 2.0 is planned to address the scenarios where the 1.x series makes deliberate simplifications that limit deployment scope. The two primary areas are: decentralised Hub architecture (enabling agents to use multiple Hubs simultaneously without federation overhead, using a distributed consensus mechanism for the AID registry and revocation index) and threshold credential issuance (requiring multiple service providers or Hub operators to co-sign a CLC, for high-security deployments where single-party credential issuance is insufficient).

Both of these features require significant cryptographic infrastructure (distributed ledgers or threshold signature schemes) that would substantially increase the implementation complexity relative to the 1.x series. The 2.0 work is therefore planned to begin only after the 1.x series has achieved broad deployment and the Working Group has had the opportunity to study operational experience from diverse deployment contexts.

Community and Ecosystem Development

Beyond the technical roadmap, the AGEX Foundation's community development goals include: the establishment of an AGEX Developer Certification programme (similar to AWS Certified Solutions Architect, for developers building AGEX-integrated systems), annual AGEX Security Summit events where Hub operators, IAs, and SPs share operational experience and security research, a bug bounty programme for security vulnerabilities in the AGEX reference implementation and specification, and a grants programme for open source AGEX tooling (SDK implementations in additional languages, monitoring dashboards, APL policy testing tools).

Appendix A: Glossary

Term	Definition
AID	Agent Identity Document. Cryptographically signed document establishing an agent's verifiable identity within the AGEX trust infrastructure.
AHFP	AGEX Hub Federation Protocol. Companion specification defining cross-Hub credential brokering for multi-organisational deployments.
APL	AGEX Policy Language. JSON-based declarative language for expressing credential approval policies evaluated by the Hub Policy Engine.
ARP	Autonomous Rotation Protocol. The AGEX sub-protocol defining agent-initiated credential rotation without human involvement.
CHA	Cross-Hub Attestation. A signed document by which one Hub attests an agent's identity to another Hub in an AHFP cross-Hub flow.
CLC	Credential Lifecycle Contract. The governing agreement defining a credential relationship between a service provider and an agent.
ERS	Emergency Revocation System. AGEX mechanism for immediate cascade revocation of an agent's complete credential set across all services.
AGEX Hub	Central infrastructure component mediating identity verification, credential brokering, rotation coordination, audit logging, and ERS.
Delegation Chain	A sequence of scoped credential sub-grants from parent to child agents, enforcing scope monotonicity at each link.

Term	Definition
Intent Manifest	Structured declaration submitted by an agent describing the purpose, scope, duration, and constraints of a credential request.
Trust Tier	A 0-3 classification reflecting verified identity level and audit history, determining degree of automated approval.
Scope Ceiling	Maximum permissions available to an agent via escalation, defined in the CLC, enforced during escalation and delegation.
Chain Provenance	Record of ancestor CLC IDs in a sub-CLC, enabling cascade revocation to all descendants of a revoked credential.
Root of Trust	The AGEX Foundation-operated Certificate Authority serving as ultimate trust anchor for the AID certificate chain.
Identity Authority	An organisation certified by the AGEX Foundation to issue AIDs to agents within its trust domain.
Credential Envelope	The ECDH-ES+AES256GCM encrypted container holding credential plaintext in a CLC, decryptable only by the beneficiary agent.

Appendix B: Notation and Conventions

This specification uses the following notation conventions throughout. Normative keywords (MUST, SHOULD, MAY, MUST NOT, SHOULD NOT) are used per RFC 2119.

Notation	Meaning
MUST / MUST NOT	Absolute requirement / prohibition per RFC 2119.
SHOULD / SHOULD NOT	Strong recommendation; deviation requires documented justification.
MAY	Optional; implementation choice.
CANONICAL_JSON(x)	RFC 8785 canonical JSON serialisation of object x.
ED25519_SIGN(m, k)	Ed25519 signature over message m using private key k, per RFC 8032.
ECDH_ES_DECRYPT(c, sk)	Decrypt credential envelope c using private key sk via ECDH-ES per RFC 8037.
ECDH_ES_ENCRYPT(m, pk)	Encrypt message m for recipient with public key pk using ECDH-ES+AES256GCM.
VERIFY_ED25519(sig, pk, m)	Verify Ed25519 signature sig over message m using public key pk. MUST raise if invalid.
UUID4()	Generate a random UUID version 4 per RFC 4122.
NOW()	Current time as UTC ISO 8601 timestamp.
SUBSET(A, B)	True if every element of set A is also in set B.
UNION(A, B)	Set of all elements in A or B.
INTERSECT(A, B)	Set of elements present in both A and B.
HSM	Hardware Security Module. Private key material SHOULD be stored in an HSM in production deployments.
	Placeholder indicating a UUID v4 value.
	Placeholder indicating a UTC timestamp in ISO 8601 format (e.g. 2026-02-18T10:00:00Z).

Notation	Meaning
	Placeholder indicating base64url-encoded bytes per RFC 4648 Section 5, without padding.

Appendix C: Change Log

Version	Date	Changes
0.1	19 February 2026	Initial public draft. All sections subject to revision based on community feedback. Security analysis has not been independently verified. Schemas are not yet frozen.

Appendix D: Wire Format Examples

This appendix provides complete, end-to-end wire format examples for each AGEX protocol flow. Examples use realistic but fictional identifiers suitable as test vectors for implementors. Cryptographic material is illustrative only. Verified test vectors with computable cryptographic material are published by the AGEX Foundation conformance suite.

D.1 Complete AID

```
-- COMPLETE AID WIRE FORMAT EXAMPLE --

{
  "aid_version": "1.0",
  "aid_id": "7b2f1e3a-4c5d-4e6f-a7b8-9c0d1e2f3a4b",
  "issuer": {
    "ia_id": "00112233-4455-6677-8899-aabbccddeeff",
    "ia_did": "did:agex:00112233-4455-6677-8899-aabbccddeeff",
    "ia_name": "AGEX Identity Authority EU",
    "ia_cert_id": "cert-2026-eu-001"
  },
  "issued_at": "2026-02-18T09:00:00Z",
  "expires_at": "2027-02-18T09:00:00Z",
  "agent": {
    "name": "AnalystBot v2.1",
    "version": "2.1.0",
    "framework": "LangChain 0.3.2",
    "type": "worker",
    "capabilities": ["data_extraction", "report_generation", "data_analysis"],
    "principal": {
      "organisation": "Acme Analytics Ltd",
      "org_id": "acme-org-f3a2b1c0-d4e5-6f7a-8b9c-0d1e2f3a4b5c",
      "contact": "agents@acme-analytics.example.com",
      "jurisdiction": "GB"
    }
  },
  "trust_tier": 1,
  "public_key": {
    "kty": "OKP",
    "crv": "Ed25519",
    "x": "11qYAYKxCrfVS_7TyWQHOg7hcvPapiMlrwIaaPcHURo"
  },
  "renewal_policy": {
    "auto_renew": true,
    "renewal_window_days": 30,
    "renewal_requires_new_key": true
  },
  "restrictions": {
    "allowed_services": ["acme-analytics-api", "acme-reports-api"],
    "denied_services": [],
    "max_clc_duration_seconds": 86400,
    "max_delegation_depth": 0,
    "geo_restrictions": ["GB", "EU"]
  },
  "ia_signature": "MEUCIQDx9pHnY...yKzLmNoPq [Ed25519 sig, base64url]"
}
```

D.2 Complete Intent Manifest

```
-- COMPLETE INTENT MANIFEST WIRE FORMAT EXAMPLE --

{
  "manifest_version": "1.0",
  "manifest_id": "a1b2c3d4-e5f6-7890-abcd-ef1234567890",
  "created_at": "2026-02-18T10:00:00Z",
  "requesting_aid": "7b2f1e3a-4c5d-4e6f-a7b8-9c0d1e2f3a4b",
  "target": {
    "service_id": "acme-analytics-api",
    "service_version": "v3",
    "resource_paths": ["/api/v3/datasets/*", "/api/v3/exports/*"],
    "requested_scopes": ["analytics:read", "exports:create"],
    "minimum_scopes": ["analytics:read"],
    "environment": "production"
  },
  "intent": {
    "summary": "Weekly KPI report generation for Q1 2026: read sales datasets, export summary. Read-only data acces
    "task_id": "task-weekly-kpi-2026-w07",
    "task_type": "read",
    "data_classification": "internal",
    "automated": true,
    "reversible": true,
    "human_visible": false,
    "estimated_api_calls": 150
  },
  "duration": {
    "estimated_start": "2026-02-18T10:00:00Z",
    "estimated_end": "2026-02-18T12:00:00Z",
    "max_duration_seconds": 7200,
    "idle_timeout_seconds": 300,
    "allow_extension": false
  },
  "data_handling": {
    "retention_policy": "session",
    "pii_processing": false,
    "pii_categories": [],
    "cross_border_transfer": false,
    "transfer_jurisdictions": [],
    "encryption_at_rest": true,
    "deletion_on_completion": true
  },
  "delegation": {
      "sub_agents_permitted": false,
      "max_delegation_depth": 0,
      "permitted_sub_agent_types": [],
      "sub_agent_scope_ceiling": []
    },
    "escalation": {
      "human_approval_thresholds": {
        "transaction_value_usd": null,
        "data_records_accessed": 10000,
        "new_resource_types": true,
        "geographic_expansion": true,
        "pii_volume_records": null
      },
      "escalation_contact": "ops@acme-analytics.example.com",
      "escalation_timeout_seconds": 3600,
      "timeout_action": "deny"
    },
    "audit": {
      "structured_logging": true,
      "log_endpoint": null,
      "reporting_interval": 3600
    },
    "agent_signature": "AAAA...AgentSig...ZZZZ [Ed25519 sig over canonical manifest]"
  }
```

D.3 Complete Credential Lifecycle Contract (CLC)

```
-- COMPLETE CLC WIRE FORMAT EXAMPLE --

{
  "clc_version": "1.0",
  "clc_id": "c0ffee01-dead-beef-1234-567890abcdef",
  "issued_at": "2026-02-18T10:00:05Z",
  "issuer_service_id": "acme-analytics-api",
  "beneficiary_aid": "7b2f1e3a-4c5d-4e6f-a7b8-9c0d1e2f3a4b",
  "manifest_id": "a1b2c3d4-e5f6-7890-abcd-ef1234567890",
  "manifest_hash": "a3f8b2c1d4e5f6a7b8c9d0e1f2a3b4c5d6e7f8a9b0c1d2e3f4a5b6c7d8e9f0a1",
  "credential_envelope": {
    "algorithm": "ECDH-ES+AES256GCM",
    "epk": {
      "kty": "OKP",
      "crv": "X25519",
      "x": "hSDwCYkwp1R0i33ctD73Wg2_Og0mOBr066SpjqqbTmo"
    },
    "ciphertext": "5eym8TW_c8SuK0ltJ3rpYIzOeDQz7TALvtu6UG9oMo4vpzs...",
    "nonce": "97239424846A3679857761FF",
    "tag": "skyYh9xbAflFOy0kfyxqeQ",
    "protected": "eyJhbGciOiJFQ0RILUVTK0FFUzI1NkdDTSIsImVuYyI6IkEyNTZHQ00ifQ"
  },
  "granted_scopes": ["analytics:read", "exports:create"],
  "scope_ceiling": ["analytics:read", "analytics:write", "exports:create", "exports:delete"],
  "denied_scopes": ["admin:all", "billing:read", "users:admin"],
  "validity": {
    "not_before": "2026-02-18T10:00:05Z",
    "not_after": "2026-02-19T10:00:05Z",
    "idle_timeout_seconds": 300,
    "geographic_restriction": ["GB", "EU"]
  },
  "rotation_policy": {
    "rotation_interval_seconds": 3600,
    "rotation_overlap_seconds": 300,
    "max_rotations": 24,
    "rotation_requires_new_manifest": false,
    "rotation_requires_human_approval": false,
    "key_derivation_function": "HKDF-SHA256"
  },
  "delegation_policy": {
    "delegation_permitted": false,
    "max_delegation_depth": 0,
      "scope_reduction_required": true,
      "permitted_sub_agent_types": [],
      "max_concurrent_sub_clcs": 0
    },
    "audit_requirements": {
      "reporting_interval_seconds": 3600,
      "log_endpoint": null,
      "anomaly_thresholds": {
        "requests_per_minute": 60,
        "error_rate_percent": 5.0,
        "unusual_hours": true
      },
      "anomaly_action": "alert"
    },
    "termination_conditions": [
      {"condition_type": "expiry",       "action": "revoke", "grace_seconds": 0},
      {"condition_type": "idle_timeout", "action": "revoke", "grace_seconds": 0},
      {"condition_type": "anomaly",      "action": "alert",  "grace_seconds": 60}
    ],
    "chain_provenance": [],
    "hub_binding": {
      "hub_id": "agex-hub-eu-west-1.example.com",
      "hub_timestamp": "2026-02-18T10:00:05Z",
      "hub_signature": "HubSig...AAAA...ZZZZ [Ed25519 sig over CLC canonical JSON]"
    },
    "provider_signature": "ProviderSig...BBBB...YYYY [Ed25519 sig over CLC excl hub_binding]"
  }
```

D.4 ARP Rotation Exchange

```
-- ARP ROTATION EXCHANGE WIRE FORMAT EXAMPLE --

Step 1: Agent -> Hub  POST /agex/v1/credentials/rotate

Headers:
  X-AGEX-Version: 1.0
  X-AGEX-Request-ID: rot-req-f1e2d3c4-b5a6-7890-abcd-ef1234567890
  X-AGEX-Timestamp: 2026-02-18T11:00:00Z
  X-AGEX-AID: 7b2f1e3a-4c5d-4e6f-a7b8-9c0d1e2f3a4b
  X-AGEX-Signature: AgentRotSig...

Body:
{
  "arp_version": "1.0",
  "message_type": "rotation_request",
  "request_id": "rot-req-f1e2d3c4-b5a6-7890-abcd-ef1234567890",
  "clc_id": "c0ffee01-dead-beef-1234-567890abcdef",
  "requesting_aid": "7b2f1e3a-4c5d-4e6f-a7b8-9c0d1e2f3a4b",
  "rotation_reason": "scheduled",
  "new_public_key": {
    "kty": "OKP", "crv": "Ed25519",
    "x": "NEW_PK_22qZBYLyCsfWU_8UzXRIOh8iduPbqjNmsJbbQdTRVSp"
  },
  "requested_validity_seconds": 86400,
  "nonce": "YWJjZGVmZ2hpamts",
  "timestamp": "2026-02-18T11:00:00Z",
  "agent_signature": "RotationReqSig..."
}

Step 2: Hub -> SP  POST <sp_agex_endpoint>/rotate

{
  "message_type": "rotation_forwarding",
  "request_id": "rot-req-f1e2d3c4-b5a6-7890-abcd-ef1234567890",
  "clc_id": "c0ffee01-dead-beef-1234-567890abcdef",
  "new_public_key": { "kty": "OKP", "crv": "Ed25519", "x": "NEW_PK_..." },
  "rotation_reason": "scheduled",
  "hub_timestamp": "2026-02-18T11:00:01Z",
  "hub_nonce": "aGlqa2xtbm9wcQ",
  "hub_signature": "HubForwardSig..."
}
```

```
Step 3: SP -> Hub  200 OK

{
  "message_type": "rotation_response",
  "request_id": "rot-req-f1e2d3c4-b5a6-7890-abcd-ef1234567890",
  "new_clc_id": "c0ffee02-dead-beef-1234-567890abcdef",
  "new_credential_envelope": { ... ECDH-ES under new_public_key ... },
  "old_clc_expiry": "2026-02-18T11:05:00Z",
  "overlap_seconds": 300,
  "provider_timestamp": "2026-02-18T11:00:02Z",
  "provider_signature": "ProviderRotSig..."
}

Step 4: Hub -> Agent  200 OK to original POST /rotate

{
  "status": "rotated",
  "new_clc": { ... full new CLC ... },
  "old_clc_expiry": "2026-02-18T11:05:00Z",
  "overlap_seconds": 300,
  "hub_timestamp": "2026-02-18T11:00:03Z"
}
```

D.5 ERS Signal and Response

```
-- ERS SIGNAL AND RESPONSE WIRE FORMAT EXAMPLE --

POST /agex/v1/ers/signal

{
  "message_type": "ers_signal",
  "signal_id": "ers-00001111-2222-3333-4444-555566667777",
  "target_aid": "7b2f1e3a-4c5d-4e6f-a7b8-9c0d1e2f3a4b",
  "reason": "compromise_suspected",
  "initiated_by": "acme-org-f3a2b1c0-d4e5-6f7a-8b9c-0d1e2f3a4b5c",
  "initiator_type": "organisation",
  "cascade": true,
  "timestamp": "2026-02-18T14:30:00Z",
  "initiator_signature": "OrgRevocationSig..."
}

Response 200 OK:

{
  "status": "revocation_initiated",
  "signal_id": "ers-00001111-2222-3333-4444-555566667777",
  "clcs_revoked": 3,
  "sub_clcs_revoked": 7,
  "services_notified": 5,
  "completion_time_ms": 8340,
  "audit_record_id": "audit-aaaa1111-2222-3333-4444-555566667777"
}

-- Hub -> SP Revocation Notification (parallel to all affected SPs) --

POST <sp_agex_endpoint>/revoke

{
  "revocation_type": "cascade_ers",
  "signal_id": "ers-00001111-2222-3333-4444-555566667777",
  "clc_ids_to_revoke": [
    "c0ffee01-dead-beef-1234-567890abcdef",
    "c0ffee02-dead-beef-1234-567890abcdef"
  ],
  "revoke_at": "2026-02-18T14:30:00Z",
  "reason": "compromise_suspected",
  "hub_signature": "HubERSSig..."
}
```

Appendix E: Trust Tier System — Full Specification

Trust Tier System - Full Specification

Tier 0 - Unverified. Default tier. No organisational verification. Consumer agents and new enterprise agents not yet verified. Requirements: valid email verified by IA, agreement to AGEX Agent Terms of Service, no prior AID revocations for policy violations. Auto-approval permissible only for read-only, public data, short-duration (<1 hour) requests from SPs whose APL policy explicitly permits it. All write, transact, and confidential-classification requests require human approval.

Tier 1 - Verified Organisation. Organisation's legal identity confirmed by IA against official government registries (Companies House for GB, SEC EDGAR for US, equivalent for other jurisdictions). Requirements: All Tier 0 plus company registration verified, registered business address verified, named technical contact verified as employed by organisation, completed IA onboarding (typically 3-5 business days). May receive auto-approval for read and read-write scopes with internal data classification during business hours.

Tier 2 - Certified Operator. Organisation has demonstrated responsible agent operation over minimum 6 months with auditable track record. Requirements: All Tier 1 plus minimum 6 months Tier 1 operation without policy violations, completed AGEX Operator Assessment, published agent operation policy reviewed by IA, demonstrated incident response capability, signed AGEX Operator Agreement including audit rights. May receive auto-approval for most credential requests including write, transact, and sensitive read operations. Recommended tier for enterprise production agents.

Tier 3 - Regulated Enterprise. Reserved for organisations in regulated industries (financial services, healthcare, critical infrastructure). Requirements: All Tier 2 plus evidence of sector-specific regulatory compliance (FCA authorisation, NHS DSP Toolkit, ICO registration as appropriate), annual IA audit of agent deployment practices, contractual obligation to notify IA within 24 hours of any security incident affecting AGEX credentials, dedicated security contact available 24/7 for IA escalation. May receive auto-approval for full available scope range.

Trust Tier Downgrade. An IA may downgrade a tier following a policy violation, security incident, or failed audit by issuing a new AID with the revised trust_tier and revoking the existing AID within 24 hours of the decision. The agent must re-request all credentials at the downgraded tier. Downgrades are recorded in the IA audit log and reported to the AGEX Foundation quarterly for aggregate monitoring.

Appendix F: Cryptographic Construction Specifications

Cryptographic Construction Specifications

Signature Algorithm: Ed25519 (RFC 8032)

Ed25519 operates on the twisted Edwards curve birationally equivalent to Curve25519. Key generation: 32 bytes from CSPRNG -> SHA-512 hash -> clamp h[0..31] per RFC 8032 (h[0] &= 248; h[31] &= 127; h[31] |= 64) -> private scalar a = clamped h[0..31], public key A = aB where B is the Ed25519 base point.

Signing: r = SHA-512(h[32..63] || message) mod l; R = rB; S = (r + SHA-512(R || A || message)) * a mod l; signature = R || S (64 bytes).

Message format for AGEX signatures: RFC 8785 canonical JSON of the signed object. For X-AGEX-Signature header: CANONICAL_JSON(body) || X-AGEX-Timestamp || X-AGEX-Request-ID.

Key Encapsulation: ECDH-ES+AES256GCM (RFC 7516, RFC 8037)

Credential envelopes use JWE JSON Serialisation. Algorithm identifiers: alg: ECDH-ES (Ephemeral Static ECDH, RFC 8037) enc: A256GCM (AES-GCM 256-bit key, 128-bit authentication tag, 96-bit nonce) crv: X25519 (Curve25519 Montgomery form, RFC 7748)

ECDH-ES key agreement procedure: 1. Generate ephemeral X25519 keypair (epk_private, epk_public). 2. Convert recipient Ed25519 public key to X25519 form per RFC 8037 Section 2: extract y-coordinate with sign bit cleared; compute u = (1+y)/(1-y) mod p where p = 2^255 - 19; encode u as 32-byte little-endian. 3. Shared secret: Z = X25519(epk_private, recipient_x25519_public). 4. Derive CEK: CEK = HKDF-SHA256(Z, salt="", info=CONCAT(alg_id, ".", enc_id), L=32). 5. Generate 96-bit random nonce N. 6. (ciphertext, tag) = AES-256-GCM-Encrypt(CEK, N, plaintext, AAD) where AAD = base64url(JWE Protected Header).

Algorithm Agility. AGEX 1.0 specifies a single suite: Ed25519, ECDH-ES+AES256GCM, SHA3-256, HKDF-SHA256. Implementations MUST NOT accept alternative algorithm identifiers.

Nonce Generation. All nonces MUST be generated from an OS-level CSPRNG. Nonce reuse under the same key causes catastrophic AES-GCM failure. Random 96-bit nonces give collision probability 2^{-32} for 2^{32} encryptions per key - acceptable given ARP produces new keys at each rotation cycle.

Hashing. SHA3-256 (FIPS 202) for Audit Ledger chain hashes. All hash comparisons MUST execute in constant time to prevent timing side channels.

Encoding. All binary cryptographic values encoded as base64url (RFC 4648 Section 5) without padding in AGEX JSON messages.

Appendix G: Complete Error Code Reference

```
Complete Error Code Reference

Authentication Errors (HTTP 401):

AID_EXPIRED: AID passed its expires_at timestamp. Agent handling: Re-request AID from IA,
  register new AID with Hub, retry.

AID_REVOKED: AID explicitly revoked by issuing IA. Agent handling: Contact IA; do not retry
  with same AID. Hub detects via revocation index updated within 60 seconds of IA notification.

AID_SIGNATURE_INVALID: ia_signature on AID failed Ed25519 verification. Agent handling:
  Request new AID from IA; report potential AID corruption. Hub verifies against IA registered key.

MANIFEST_SIGNATURE_INVALID: agent_signature on Intent Manifest failed verification.
  Agent handling: Verify signing logic; check canonical JSON is correctly formed.

REQUEST_SIGNATURE_INVALID: X-AGEX-Signature header failed verification.
  Agent handling: Verify request signing implementation and canonical message format.

Request Errors (HTTP 400):

INVALID_MANIFEST: Intent Manifest failed JSON schema validation. Response includes
  details.field and details.reason. Agent handling: Fix manifest; consult Section 4.2 schema.

SERVICE_NOT_FOUND: target.service_id not registered with this Hub. Agent handling: Verify
  service_id; use discovery endpoint to find correct Hub.

SCOPE_NOT_AVAILABLE: One or more requested_scopes not offered by target service.
  Response includes details.unavailable_scopes. Agent handling: Consult service discovery.

TIMESTAMP_OUT_OF_RANGE: X-AGEX-Timestamp more than 300 seconds from Hub current time.
  Agent handling: Synchronise system clock; use reliable NTP source.

DUPLICATE_REQUEST_ID: X-AGEX-Request-ID seen within 24-hour deduplication window.
  Agent handling: Always generate fresh UUID4 for each request.

UNSUPPORTED_VERSION: X-AGEX-Version specifies unsupported Hub version. Response includes
  details.supported_versions.

Authorisation Errors (HTTP 403):

REQUEST_REJECTED: Credential request rejected by APL policy engine. Response includes
```

```
    details.policy_rule (rule_id) and details.reason. Agent handling: Review APL policy with SP.

  SCOPE_CEILING_EXCEEDED: Escalation or delegation request includes scopes exceeding scope_ceiling.
    Agent handling: Request only scopes within the defined ceiling.

  INITIATOR_NOT_AUTHORISED: ERS signal from party not authorised to revoke target AID.
    Authorised parties: agent's organisation, any SP with active CLCs, Hub operator, issuing IA.

  Conflict Errors (HTTP 409):

  ROTATION_IN_PROGRESS: Rotation for specified CLC already in progress. Response includes
    details.existing_request_id. Agent handling: Wait for existing rotation; poll if needed.

  AID_ALREADY_REGISTERED: aid_id already registered with this Hub. Do not re-register.

  APPROVAL_NOT_FOUND: approval_token not found or expired. Agent handling: Re-submit credential
    request.

  Rate Limiting (HTTP 429):

  RATE_LIMIT_EXCEEDED: Request rate limit exceeded. Response includes Retry-After header and
    details.limit_type (aid|org|ip|global). Agent handling: Exponential backoff; respect
    Retry-After header.

  Server Errors (HTTP 5xx):

  SP_UNAVAILABLE (503): Target SP did not respond within 30 seconds.
    Agent handling: Retry with exponential backoff.

  SP_ERROR (502): Target SP returned an error response. Response includes details.sp_error.
    Agent handling: Consult SP documentation; contact SP support if persistent.

  HUB_INTERNAL_ERROR (500): Internal Hub error. Response includes support_id.
    Agent handling: Retry after delay; report support_id to Hub operator if persistent.

  ERS_TIMEOUT (504): ERS cascade revocation did not complete within 60-second bound.
    This is a Hub conformance violation. Agent handling: Treat all credentials as potentially
    unrevoked; contact Hub operator immediately.
```

Appendix H: Interoperability Guidance

Interoperability Guidance

The AGEX Foundation operates an interoperability testing environment at interop.agex.api including a reference Hub implementation, reference agent stubs for SP testing, and reference SP stubs for agent and Hub testing. Conformance certification requires demonstrated interoperability with all mandatory conformance requirements.

Integration with Existing Systems

From static API keys: Map static keys to CLCs with long validity periods and rotation disabled initially. Enable ARP incrementally as operational confidence grows. The parallel-credential approach (keep static key active during transition) is supported.

From OAuth 2.0 client credentials: Map client_id/client_secret pairs to AID/CLC pairs. Scope string format is intentionally compatible with OAuth 2.0 scope strings. Service providers can issue parallel OAuth and AGEX credentials during migration.

From SPIFFE/SPIRE: SPIFFE SVIDs may be included in AID capabilities as supplementary claims. Hub implementations may accept SPIFFE SVIDs in X-AGEX-SPIFFE-SVID headers for hybrid deployments.

Hub Implementation Architectural Notes

Zero-knowledge requirement (HC-M-005): The Hub database schema must never contain plaintext credential material columns. Schema migrations must be reviewed for this.

ERS 60-second bound (HC-M-008): SP notification and Hub database revocation must be parallelised, not sequential. Sequential implementation cannot reliably meet the bound for large delegation chains across many service providers.

Tamper-evident audit ledger (HC-M-009): Distributed Hub deployments must designate a single authoritative write path for the audit ledger to prevent hash chain splits.

24-hour deduplication window (HC-M-010): Requires a distributed cache (Redis, Memcached) accessible to all Hub instances. Local in-memory deduplication is not conformant for multi-instance deployments.

Appendix I: Future Work

Future Work

Priority Tier 1 - Version 1.1 (Target Q4 2026):

FW-1.1 Verifiable Presentations Integration: Define a VC profile for AIDs enabling agents to present AGEX credentials in W3C VC format, unlocking interoperability with EU eIDAS 2.0.

FW-1.2 Agent Capability Endorsements: Define a standard vocabulary of agent capability claims and third-party endorsement mechanism, enabling SP APL policies to require endorsed capabilities (e.g. PCI-DSS certified, HIPAA trained).

FW-1.3 Multi-Party Authorisation: Define a K-of-N human approval extension to the APL for high-stakes operations requiring simultaneous authorisation from multiple principals.

FW-1.4 Streaming Credentials: Define a Streaming Credential extension for agents using WebSocket, gRPC streaming, or SSE connections where the CLC model does not map directly.

Priority Tier 2 - Version 1.2 (Target H1 2027):

FW-2.1 Agent Reputation System: Dynamic trust signals based on observed behaviour, rewarding policy compliance and penalising anomalous usage, available via a reputation query endpoint as an optional overlay on the static tier system.

FW-2.2 Hardware-Backed Agent Identity: Optional hardware attestation extension to AIDs, allowing agents to prove private key material is held in a TPM or secure enclave using TPM 2.0 Quote, AMD SEV attestation, or AWS Nitro attestation.

FW-2.3 Batch Credential Operations: Batch endpoint for orchestrators to acquire credentials for hundreds of worker agents simultaneously in a single Hub API call with atomic semantics.

FW-2.4 Agent-to-Agent Direct Channels: Direct authentication channel for two agents to authenticate each other using their AIDs without Hub mediation, for latency-sensitive agent-to-agent interactions.

Research Items (Post-1.2):

FW-3.1 Privacy-Preserving Audit: Zero-knowledge proof techniques (Bulletproofs, zk-STARKs) to enable audit records proving policy compliance without revealing agent identity.

FW-3.2 Post-Quantum Cryptography: Migration path to NIST-standardised post-quantum algorithms (ML-DSA for signatures, ML-KEM for key encapsulation) with hybrid transition period.

FW-3.3 Cross-Protocol Bridges: Bridge specifications enabling AGEX credentials to be presented to non-AGEX services (SPIFFE, mTLS, OAuth 2.0) via protocol adapters.

Acknowledgements

Acknowledgements

The authors thank the SPIFFE/SPIRE community at CNCF for pioneering workload identity infrastructure whose insights directly shaped the ARP design; the IETF OAuth working group for the Rich Authorisation Requests extension that informed Intent Manifest design; the W3C Verifiable Credentials working group for DID and VC primitives underpinning AGEX identity; and the security research community whose work on credential management vulnerabilities in cloud-native systems provided the empirical basis for the threat model.

The AGEX specification was developed through extensive dialogue with AI agent framework developers, API service providers, enterprise security architects, and legal practitioners in the GDPR and AI governance space.

This work was supported by the AGEX Foundation.