IntentID Protocol Specification

Open Specification — Version 0.2 Draft

Author: Vivek Chakravarthy Durairaj, Founder & CEO, Cogumi, Inc.

Date: February 2026

License: Apache 2.0

Status: Draft — Open for Community Review

Repository: github.com/cogumi/intentid-spec

Website: intentid.org

Abstract

IntentID is an open protocol that introduces declared intent as a cryptographic primitive in AI agent identity. An agent's identity is the composite of OrgID (optional) + UserID (human principal) + IntentID (hash of a signed Intent Contract declaring purpose, tool manifest, scope, and constraints). Any modification to the agent invalidates its IntentID, requiring re-authorization. Version 0.2 adds: Contract Revocation (CRL + live status endpoint), Model Attestation Object (supporting both self-hosted and API-based deployments), Key Rotation with versioned kid fields, Intent Coherence Checking with a dual-layer taxonomy (IntentID reference taxonomy + enterprise custom extension), Action Sequence Constraints for multi-step confused deputy prevention, and three Compliance Tiers (Individual, Professional, Enterprise). This specification defines the complete protocol including Intent Contract schema, AgentID construction, revocation, delegation chain rules, verification gate algorithm, and compliance requirements. It is submitted as a response to the NIST NCCoE concept paper on AI Agent Identity and Authorization (February 2026) and proposed for standardization through the OpenID Foundation.

Document Status

This is Version 0.2 of the IntentID specification. It supersedes Version 0.1 (February 2026). Changes in 0.2: Contract Revocation (Section 3.5), Model Attestation Object (Section 4.2.3), Key Rotation (Section 4.5), Intent Coherence Checking (Section 6.3), Action Sequence Constraints (Section 4.6 + 6.4), Compliance Tiers (Section 9). Licensed Apache 2.0. Submit feedback via GitHub Issues at github.com/cogumi/intentid-spec.

1. Introduction and Motivation

1.1 Problem Statement

Every current AI agent identity system answers the question: 'who is this agent?' None answer the question that actually matters for security: 'what is this agent authorized to do right now, and is it still doing that?'

The fundamental flaw in existing approaches is treating agent identity the same as human or machine identity. An agent is not a person and not a static service account. It is a mutable, goal-directed system whose risk profile changes completely when its instructions change. A credential-based identity that persists across system prompt changes creates a gap an attacker can drive a truck through.

1.2 The IntentID Thesis

IntentID's core thesis: agent identity must be cryptographically bound to declared purpose. Change the purpose, change the identity. This single design choice eliminates the agent mutability problem, enables cryptographic delegation chains, and provides the foundation for zero-trust authorization at the action level.

1.3 Design Principles

- Purpose-native: intent is a first-class identity primitive, not an afterthought
- Immutability: any change to agent configuration produces a new identity requiring fresh authorization
- Human accountability: every AgentID traces back to a verifiable human principal (UserID)
- Delegation safety: permissions can only narrow downstream, never widen
- Zero-trust: every action is verified against the intent contract in real time
- Open and interoperable: works across all clouds, models, and agent frameworks
- Minimal footprint: a developer can add IntentID to any agent in under 10 lines of code

1.4 Relationship to Existing Standards

IntentID is designed to complement, not replace, existing identity infrastructure. The UserID component can reference any existing identity provider (Entra ID, Okta, Ping, AWS IAM). Intent Contracts can be encoded as JWT extensions. The verification protocol is compatible with MCP, A2A, and existing agent framework architectures. IntentID adds the intent layer that no existing standard provides.

2. Terminology and Conventions

The key words MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD NOT, RECOMMENDED, MAY, and OPTIONAL in this document are to be interpreted as described in RFC 2119.

Term	Type	Req	Definition
AgentID	string	REQUIRED	Composite identity: OrgID (opt) + UserID + IntentID
OrgID	string	OPTIONAL	Organization-scoped identifier for enterprise deployments
UserID	string	REQUIRED	Verified human principal. Always present. Accountability anchor.
IntentID	string	REQUIRED	SHA-256 hash of the canonicalized Intent Contract
Intent Contract	object	REQUIRED	Signed declaration of agent purpose, tools, scope, and constraints
Tool Manifest	array	REQUIRED	Explicit list of permitted tools and their allowed actions
Delegation Chain	array	CONDITIONAL	Ordered list of parent AgentIDs. Required for delegated agents.
Verification Gate	function	REQUIRED	Real-time authorization check on every tool invocation
Intent Coherence	property	REQUIRED	The property of actions being consistent with declared purpose
Confused Deputy	attack	—	Manipulation of a legitimate agent into unauthorized actions

3. AgentID Construction

3.1 Format

An AgentID is a colon-delimited string with the following structure:

```
AgentID = 'agent:' [ org_id ':' ] user_id ':' intent_id

Where:
  org_id    = url-encoded organization identifier (OPTIONAL)
  user_id   = url-encoded verified human principal identifier (REQUIRED)
  intent_id = 'intentid:v1:' hex(sha256(canonicalized_contract))

Examples:
  // With org:
  agent:acme_corp:john.doe%40acme.com:intentid:v1:a3f9c2b1...

  // Without org (individual):
  agent:john.doe%40acme.com:intentid:v1:a3f9c2b1...
```

3.2 OrgID

OrgID is OPTIONAL. When present it MUST be a stable, unique identifier for the organization. It SHOULD be a domain name, UUID, or other globally unique identifier. It MUST NOT contain colons or whitespace unless URL-encoded.

3.3 UserID

UserID is REQUIRED. It MUST reference a verifiable identity in an external identity provider. It SHOULD be an email address, DID (Decentralized Identifier), or provider-qualified identifier. The UserID establishes human accountability for all actions taken under this AgentID.

3.4 IntentID Computation

The IntentID MUST be computed as follows:

```
function compute_intent_id(contract):
  // 1. Create a copy; remove mutable fields
  c = deep_copy(contract)
  delete c['signature']
  delete c['intent_id']
  // 2. Canonicalize: sort all keys recursively, no whitespace
  canonical = json_canonical_serialize(c)  // RFC 8785 JCS compliant

  // 3. Hash
  return 'intentid:v1:' + hex(sha256(utf8_encode(canonical)))
```

REQUIRED
Implementations MUST use RFC 8785 JSON Canonicalization Scheme (JCS) for the canonical serialization step to ensure interoperability across implementations and languages.

3.5 Contract Revocation

IntentID provides two complementary mechanisms for revoking a contract before its not_after expiry: a Contract Revocation List (CRL) for planned revocation, and a live status endpoint for real-time revocation checks.

3.5.1 Contract Revocation List (CRL)

Each organization or user MAY maintain a CRL — an append-only, signed list of revoked IntentIDs. The CRL MUST be signed by the same key that signed the revoked contracts. Implementations operating at Professional or Enterprise tier MUST check the CRL before accepting any contract.

```
CRL entry format:
{
  "revoked_intent_id":  string,   // the IntentID being revoked
  "revocation_time":    string,   // ISO 8601 UTC timestamp
  "reason":             string,   // 'key_compromise' | 'superseded' |
                                  // 'affiliation_changed' | 'unspecified'
  "revoked_by":         string,   // UserID of the revoking principal
  "signature":          string    // ed25519 signature of this entry
}

// Revocation authorization rule:
// ONLY the UserID that signed the contract may revoke it,
// OR a designated org-level revocation authority (org_id:revocation_authority).
// Revocation by any other party MUST be rejected.
```

3.5.2 Live Status Endpoint (OCSP-style)

For real-time revocation checking, implementations SHOULD expose a live status endpoint. The verification gate MAY query this endpoint as part of contract verification. The endpoint MUST respond within 500ms or the gate MUST fall back to CRL-only verification.

```
GET /intentid/v1/status/{intent_id}

Response:
{
  "intent_id":      string,    // echoed from request
  "status":         string,    // 'valid' | 'revoked' | 'unknown'
  "checked_at":     string,    // ISO 8601 UTC
  "revoked_at":     string | null,
  "reason":         string | null,
  "next_update":    string,    // ISO 8601 — cache TTL hint
  "signature":      string     // endpoint operator signature
}
```

Security Note

Revocation entries are irrevocable. Once an IntentID is added to a CRL, it MUST NOT be removed. The not_after expiry is the only mechanism for natural contract termination. Revocation is for early termination only.

4. Intent Contract Specification

The Intent Contract is the machine-readable, cryptographically signed declaration of what an agent is, what it is authorized to do, and under what constraints. It is the source of truth for all authorization decisions.

4.1 Schema

```
{
  // === IDENTITY FIELDS ===
  "org_id":               string | null,      // OPTIONAL: organization identifier
  "user_id":              string,             // REQUIRED: human principal
  "parent_agent_id":      string | null,      // CONDITIONAL: if spawned by agent

  // === PURPOSE DECLARATION ===
  "declared_purpose":     string,             // REQUIRED: natural language purpose
  "goal_structure":       GoalStructure,      // REQUIRED: structured goal definition

  // === MODEL ATTESTATION ===
  "model_attestation":    ModelAttestation,   // REQUIRED: model identity + integrity
  "system_prompt_hash":   string,             // REQUIRED: hex(sha256(system_prompt))

  // === TOOL MANIFEST ===
  "tool_manifest": [
    {
      "tool_id":            string,           // REQUIRED: unique tool identifier
      "allowed_actions":    string[],         // REQUIRED: permitted action verbs
      "data_scope":         string,           // REQUIRED: data access boundary
      "rate_limit":         RateLimit,        // REQUIRED: max call frequencies
      "conditions":         string | null     // OPTIONAL: precondition expression
    }
  ],

  // === ACTION SEQUENCE CONSTRAINTS ===
  "sequence_rules":       SequenceRule[],     // REQUIRED (empty array if none)

  // === SCOPE CONSTRAINTS ===
  "data_classification":  string[],           // REQUIRED: permitted data classes
  "output_restrictions":  object,             // REQUIRED: output destination rules
  "escalation_triggers":  EscalationRule[], // REQUIRED: human escalation
conditions

  // === TEMPORAL BOUNDS ===
  "not_before":           string,             // REQUIRED: ISO 8601 datetime
  "not_after":            string,             // REQUIRED: ISO 8601 datetime

  // === CRYPTOGRAPHIC FIELDS (set at signing) ===
  "issued_at":            string,             // REQUIRED: ISO 8601 signing time
  "kid":                  string,            // REQUIRED: signing key identifier
  "signature":            string,            // REQUIRED:
base64url(ed25519_sign(canonical))
  "intent_id":            string             // REQUIRED: computed per Section 3.4
}
```

4.2 Field Specifications

4.2.1 declared_purpose

A human-readable string describing the agent's authorized purpose. MUST be specific enough to enable intent coherence checking. MUST NOT be a generic description such as 'general assistant'. SHOULD describe the domain, task type, and any critical constraints in plain language.

4.2.2 goal_structure

A structured object enabling machine-readable intent classification. The goal_structure is the anchor for intent coherence checking (Section 6.3). It MUST include at minimum the following fields drawn from the IntentID Reference Taxonomy (Appendix A). Enterprise deployments MAY extend this with custom_taxonomy (Section 6.3.2).

```
{
  "type":              string,    // REQUIRED. From taxonomy: 'task_completion' |
                                 // 'monitoring' | 'transformation' | 'retrieval' |
                                 // 'communication' | 'execution' | 'analysis'
  "domain":            string,    // REQUIRED. From taxonomy: 'software_development'
|
                                 // 'customer_support' | 'finance' | 'legal' |
                                 // 'hr' | 'it_operations' | 'data_engineering' |
                                 // 'security' | 'content_creation' | 'research'
                                 // (or custom domain — see Section 6.3.2)
  "scope":             string,    // REQUIRED. From taxonomy: 'read_only' |
                                 // 'read_write' | 'execute' | 'communicate'
  "targets":           string[],  // REQUIRED. The specific resources or artifacts
                                 // this agent operates on. e.g. ['source_code',
                                 // 'pull_requests', 'test_suites']
  "forbidden_domains": string[], // REQUIRED (empty array if none). Explicit list
                                 // of taxonomy domains this agent must never enter.
                                 // e.g. ['finance', 'hr', 'communication']
  "max_delegation_depth": integer, // REQUIRED. Max child agent chain depth.
Default 3.
  "custom_taxonomy":  object | null // OPTIONAL. Enterprise extension — see 6.3.2
}
```

Example — coding agent goal_structure:

```
{
  "type":               "task_completion",
  "domain":             "software_development",
  "scope":              "read_write",
  "targets":            ["source_code", "pull_requests", "test_suites"],
  "forbidden_domains": ["finance", "hr", "communication", "legal"],
  "max_delegation_depth": 2,
  "custom_taxonomy":   null
}
```

4.2.3 model_attestation

The model_attestation object establishes the cryptographic identity and integrity of the model powering the agent. It replaces the simple model_hash field from v0.1 to support both self-hosted and API-based deployments. Two modes are defined:

```
ModelAttestation = {
  "mode":              string,    // REQUIRED: 'self_hosted' | 'api_hosted'
  "model_id":          string,    // REQUIRED: human-readable model identifier
                                // e.g. 'claude-sonnet-4-6', 'gpt-4o-2025-01'

  // --- Self-hosted mode fields ---
  "model_hash":        string | null,  // sha256(model_weights_bytes) in hex
  "weights_uri":       string | null,  // URI where weights can be verified

  // --- API-hosted mode fields ---
  "provider":          string | null,  // e.g. 'anthropic' | 'openai' | 'google'
  "provider_attestation": object | null, // Signed attestation from provider
  // Provider attestation schema:
  // {
  //   "model_version":    string,  // provider's internal version identifier
  //   "snapshot_date":    string,  // ISO 8601 — model snapshot this refers to
  //   "attestation_id":   string,  // provider-issued unique attestation ID
  //   "issued_at":        string,  // ISO 8601
  //   "provider_sig":     string   // provider's Ed25519 signature
  // }

  // --- Common field ---
  "system_prompt_hash": string   // REQUIRED in both modes:
                                  // hex(sha256(utf8_encode(system_prompt)))
}
```

API-Hosted Deployments

For API-hosted models, the provider_attestation is obtained from the model provider's attestation API at agent instantiation time. Anthropic, OpenAI, and other providers are expected to implement attestation endpoints as part of IntentID ecosystem adoption. Until provider attestation APIs are available, API-hosted deployments MUST at minimum record the model_id and set mode='api_hosted' with a null provider_attestation — this is valid for Individual and Professional tiers. Enterprise tier REQUIRES a valid provider_attestation.

4.2.4 tool_manifest

An explicit enumeration of every tool the agent is permitted to invoke. Implicit or wildcard tool access is NOT permitted. Each entry MUST specify allowed_actions as an explicit list — no wildcards. The rate_limit object MUST include at minimum calls_per_minute and calls_per_day.

4.2.5 output_restrictions

Constraints on where the agent may send data. MUST be specified. An empty object {} is a valid value meaning no restrictions, but MUST be explicitly set. Implementations SHOULD enforce at minimum: no_external_domains (bool), allowed_recipients (list), max_payload_size (bytes).

4.2.6 escalation_triggers

Conditions under which the agent MUST pause and notify the human principal before proceeding. MUST be specified. Each trigger MUST include: pattern (the condition), action (pause | block | notify), and notify_target (user_id or role).

4.3 Signing the Intent Contract

```
function sign_intent_contract(contract, private_key_ed25519):
  // Ensure signature and intent_id fields are absent
  assert 'signature' not in contract
  assert 'intent_id' not in contract

  // Set issued_at
  contract['issued_at'] = utc_now_iso8601()

  // Compute canonical form
  canonical = json_canonical_serialize(contract)  // RFC 8785

  // Sign
  sig_bytes = ed25519_sign(private_key_ed25519, utf8_encode(canonical))
  contract['signature'] = base64url_encode(sig_bytes)

  // Compute and attach IntentID
  contract['intent_id'] = compute_intent_id(contract)

  return contract
```

4.4 Verifying an Intent Contract

```
function verify_intent_contract(contract, public_key_ed25519):
  // Step 1: Verify IntentID
  computed_id = compute_intent_id(contract)
  if computed_id != contract['intent_id']:
    raise IntegrityError('intent_id mismatch — contract tampered')

  // Step 2: Reconstruct canonical form (without sig + intent_id)
  c = deep_copy(contract)
  c_issued_at = c['issued_at']  // preserve
  delete c['signature']
  delete c['intent_id']
  canonical = json_canonical_serialize(c)

  // Step 3: Look up public key by kid
  public_key = key_registry.lookup(contract['user_id'], contract['kid'])
  if public_key is null:
    raise KeyError(f'unknown kid {contract["kid"]} for user {contract["user_id"]}')

  // Step 4: Verify signature
  sig_bytes = base64url_decode(contract['signature'])
  if not ed25519_verify(public_key, utf8_encode(canonical), sig_bytes):
    raise SignatureError('invalid signature')

  // Step 5: Check revocation
  if crl.is_revoked(contract['intent_id']):
    raise RevocationError('contract has been revoked')

  // Step 6: Check temporal validity
  now = utc_now()
  if now < parse_iso8601(contract['not_before']):
    raise TemporalError('contract not yet valid')
  if now > parse_iso8601(contract['not_after']):
    raise TemporalError('contract expired')

  return True
```

4.5 Key Rotation

Each UserID maintains a key history in the registry — a set of public keys identified by kid (key ID). Key rotation MUST follow the overlap window pattern to avoid invalidating active contracts:

```
Key lifecycle states:
  'active'    — can sign new contracts AND verify existing ones
  'retiring'  — cannot sign new contracts, can still verify existing ones
  'revoked'   — cannot sign or verify (only if key was compromised)

Rotation procedure:
  1. Generate new key pair (new_kid)
  2. Register new public key in key registry with status='active'
  3. Set old key status to 'retiring' — it remains valid for verification
```

```
     of contracts that reference old_kid until their not_after expires
  4. All new Intent Contracts MUST use new_kid
  5. Old key transitions to 'revoked' only if compromised;
     otherwise it naturally expires when no active contracts reference it

Key registry entry:
{
  "user_id":      string,   // owner
  "kid":          string,   // key identifier (UUID or hash of public key)
  "public_key":   string,   // base64url-encoded Ed25519 public key
  "status":       string,   // 'active' | 'retiring' | 'revoked'
  "created_at":   string,   // ISO 8601
  "retired_at":   string | null,
  "revoked_at":   string | null
}
```

Key Compromise
If a private key is compromised, the key MUST be immediately set to 'revoked' status AND all Intent Contracts signed with that kid MUST be added to the CRL, regardless of their not_after date. This is the only case where revocation of multiple contracts in a single operation is permitted.

4.6 Action Sequence Constraints

Action Sequence Constraints prevent multi-step confused deputy attacks — scenarios where each individual tool call is within scope, but the sequence of calls collectively constitutes unauthorized behavior. The verification gate maintains a rolling action window per agent session and evaluates each new action against the declared sequence rules.

A sequence rule declares a forbidden combination of actions within a single task context:

```
SequenceRule = {
  "rule_id":      string,    // REQUIRED: unique rule identifier
  "description":  string,    // REQUIRED: human-readable explanation
  "pattern":      string[],  // REQUIRED: ordered list of tool_id:action pairs
                             // that are forbidden in sequence
  "window":       integer,   // REQUIRED: number of recent actions to consider
  "on_match":     string,    // REQUIRED: 'block' | 'escalate'
  "unless":       string | null  // OPTIONAL: condition that exempts this rule
}

Example — prevent data exfiltration via read+email sequence:
{
  "rule_id":     "no-read-then-email",
  "description": "Prevent reading sensitive files then sending external email",
  "pattern":     ["filesystem:read", "email:send_external"],
  "window":      10,
  "on_match":    "block",
  "unless":      "email.recipient in
contract.output_restrictions.allowed_recipients"
}

Example — require human approval before executing database writes:
{
  "rule_id":     "db-write-after-read-requires-approval",
  "description": "Any db:write following db:read requires escalation",
  "pattern":     ["database:read", "database:write"],
  "window":      5,
  "on_match":    "escalate",
  "unless":      null
}
```

4.6.1 Sequence Rule Evaluation Algorithm

```
function check_sequence_rules(agent_id, tool_id, action, contract, session_window):
  // session_window is a rolling list of recent (tool_id:action) pairs
  candidate = f'{tool_id}:{action}'

  for rule in contract.sequence_rules:
    pattern = rule.pattern
    window = rule.window
    unless = rule.unless

    // Get recent actions within window, append candidate
    recent = session_window.last(window - 1) + [candidate]

    // Check if pattern appears as a subsequence of recent
    if is_subsequence(pattern, recent):
      // Check unless condition
      if unless and evaluate_condition(unless, context):
        continue  // exempted
      if rule.on_match == 'block':
        return DENY(f'sequence_rule_violated:{rule.rule_id}')
      elif rule.on_match == 'escalate':
        return ESCALATE(rule, notify=contract.user_id)

  return CONTINUE  // no sequence rules triggered
```

5. Delegation Chain

When an agent spawns a child agent, it issues a delegated Intent Contract. The delegation chain preserves human accountability through arbitrary agent depth while enforcing that permissions can only narrow, never widen.

5.1 Delegation Rules

The following rules are REQUIRED and MUST be enforced by all compliant implementations:

- RULE 1 — Principal Preservation: child.user_id MUST equal parent.user_id. child.org_id MUST equal parent.org_id if present.
- RULE 2 — Scope Narrowing: child.tool_manifest MUST be a subset of parent.tool_manifest. For each tool, child.allowed_actions MUST be a subset of parent.allowed_actions. child rate limits MUST be equal to or less than parent rate limits.
- RULE 3 — Temporal Containment: child.not_before MUST be >= parent.not_before. child.not_after MUST be <= parent.not_after.
- RULE 4 — Parent Reference Integrity: child.parent_agent_id MUST equal the AgentID of the parent, computed per Section 3.
- RULE 5 — Depth Limit: the maximum delegation depth MUST be declared in the root contract's goal_structure.max_delegation_depth. If not declared, default is 3. Implementations MUST reject chains exceeding this depth.

5.2 Delegation Chain Validation Algorithm

```
function validate_delegation_chain(child_contract, parent_contract):

  // Rule 1: Principal preservation
  assert child_contract.user_id == parent_contract.user_id
  if parent_contract.org_id:
    assert child_contract.org_id == parent_contract.org_id

  // Rule 2: Scope narrowing
  parent_tool_map = {t.tool_id: t for t in parent_contract.tool_manifest}
  for child_tool in child_contract.tool_manifest:
    if child_tool.tool_id not in parent_tool_map:
      raise DelegationError(f'tool {child_tool.tool_id} not in parent manifest')
    parent_tool = parent_tool_map[child_tool.tool_id]
    if not set(child_tool.allowed_actions) <= set(parent_tool.allowed_actions):
      raise DelegationError('child actions exceed parent permissions')
    if child_tool.rate_limit.calls_per_minute >
parent_tool.rate_limit.calls_per_minute:
      raise DelegationError('child rate limit exceeds parent')

  // Rule 3: Temporal containment
```

```
assert child_contract.not_before >= parent_contract.not_before
assert child_contract.not_after <= parent_contract.not_after

// Rule 4: Parent reference integrity
expected_parent_id = construct_agent_id(
  parent_contract.org_id, parent_contract.user_id, parent_contract
)
assert child_contract.parent_agent_id == expected_parent_id

return True
```

6. Verification Gate

The Verification Gate is the real-time authorization check that MUST be applied before every tool invocation. It is the enforcement point of the IntentID protocol.

6.1 Gate Algorithm

```
function verification_gate(agent_id, tool_id, action, data_ref, output_dest,
session):
  timestamp = utc_now()

  // 1. Resolve and verify contract (includes revocation + temporal checks)
  contract = registry.resolve(agent_id)
  verify_intent_contract(contract, key_registry.lookup(contract.user_id,
contract.kid))

  // 2. Tool authorization
  tool = find(contract.tool_manifest, tool_id=tool_id)
  if not tool:
    return DENY('tool_not_in_manifest')

  // 3. Action authorization
  if action not in tool.allowed_actions:
    return DENY('action_not_permitted')

  // 4. Data scope
  if not data_ref.within_scope(tool.data_scope):
    return DENY('data_out_of_scope')

  // 5. Output restriction
  if not output_dest.satisfies(contract.output_restrictions):
    return DENY('output_restricted')

  // 6. Rate compliance
  if rate_exceeded(agent_id, tool_id, tool.rate_limit, timestamp):
    return DENY('rate_limit_exceeded')

  // 7. Intent coherence check (Enterprise tier)
  coherence = check_intent_coherence(tool_id, action, contract.goal_structure)
  if coherence.distance > coherence.threshold:
    return ESCALATE('intent_coherence_anomaly', notify=contract.user_id)

  // 8. Action sequence constraints
  seq_result = check_sequence_rules(agent_id, tool_id, action, contract,
session.window)
  if seq_result != CONTINUE:
    return seq_result  // DENY or ESCALATE from sequence rule

  // 9. Escalation triggers
  for trigger in contract.escalation_triggers:
    if trigger.matches(action, data_ref):
      return ESCALATE(trigger, notify=contract.user_id)

  // 10. Delegation chain (if delegated)
  if contract.parent_agent_id:
    parent = registry.resolve(contract.parent_agent_id)
    validate_delegation_chain(contract, parent)

  // 11. Update session window and log
  session.window.append(f'{tool_id}:{action}')
  audit_log.append({
    agent_id, tool_id, action, data_ref, output_dest,
    timestamp, intent_id: contract.intent_id,
    user_id: contract.user_id, kid: contract.kid
  })
  return ALLOW
```

6.2 Audit Log Requirements

Every ALLOW decision MUST produce an audit log entry. Every DENY and ESCALATE decision MUST produce an audit log entry with the denial reason. Audit log entries MUST be append-only and tamper-evident. Implementations SHOULD use a Merkle-tree or blockchain-style structure for the audit log to provide cryptographic proof of completeness. Enterprise tier REQUIRES tamper-evident audit logs with cryptographic chaining.

6.3 Intent Coherence Checking

Intent coherence checking is the runtime enforcement of the goal_structure declaration. It detects when an agent is being manipulated into actions that are individually authorized (present in the tool manifest) but semantically outside the agent's declared domain and purpose.

6.3.1 IntentID Reference Taxonomy

IntentID defines a reference taxonomy of agent domains, action types, and tool categories. Each tool registration includes a semantic_category drawn from this taxonomy. The coherence check computes the semantic distance between a proposed tool call's category and the agent's declared goal_structure.domain.

```
IntentID Reference Taxonomy v1.0:

DOMAINS (goal_structure.domain values):
  software_development  → tools: code_editor, vcs, ci_cd, debugger, test_runner
  customer_support      → tools: ticket_system, crm, knowledge_base, chat
  finance               → tools: accounting, payment, banking, reporting
  legal                 → tools: contract_mgmt, compliance, document_review
  hr                    → tools: hris, payroll, recruiting, scheduling
  it_operations         → tools: monitoring, deployment, cloud_mgmt, access_mgmt
  data_engineering      → tools: database, etl, data_warehouse, ml_pipeline
  security              → tools: siem, scanner, firewall, identity_mgmt
  content_creation      → tools: cms, editor, media, publishing
  research              → tools: search, document_store, citation_mgr, notebook

CROSS-DOMAIN DISTANCE MATRIX (excerpt):
  software_development ↔ data_engineering:  distance = 0.2  (low — adjacent)
  software_development ↔ it_operations:     distance = 0.3  (low — adjacent)
  software_development ↔ finance:           distance = 0.8  (high — distinct)
  software_development ↔ hr:                distance = 0.9  (high — distinct)
  customer_support     ↔ crm tools:         distance = 0.1  (very low — core)
  customer_support     ↔ payment tools:     distance = 0.7  (high — adjacent risk)

DEFAULT COHERENCE THRESHOLD: 0.6
  Actions with semantic distance > 0.6 from declared domain trigger ESCALATE.
  Actions crossing into a declared forbidden_domain always trigger ESCALATE
  regardless of distance score.
```

6.3.2 Enterprise Custom Taxonomy

Enterprise deployments MAY define a custom taxonomy to handle novel or proprietary domains not covered by the reference taxonomy. A custom taxonomy MUST be declared in the goal_structure.custom_taxonomy field and MUST include: domain definitions, tool-to-domain mappings, and a cross-domain distance matrix. Custom taxonomies are evaluated after the reference taxonomy — if a tool is found in the custom taxonomy, the custom distance is used; otherwise the reference taxonomy applies.

```
custom_taxonomy schema:
{
  "version":    string,    // semantic version of this custom taxonomy
  "domains": [
    {
      "domain_id":    string,   // unique domain identifier
      "description":  string,   // human-readable domain description
      "tool_ids":     string[]  // tool_ids belonging to this domain
    }
  ],
  "distance_matrix": {
    // Map of 'domain_a:domain_b' -> float (0.0 to 1.0)
    // 0.0 = same domain, 1.0 = maximally distant
    // e.g. 'genomics:clinical_trial': 0.2
  },
  "coherence_threshold": float  // override default 0.6 if needed
}
```

7. Transport and Encoding

7.1 JWT Encoding

An Intent Contract MAY be encoded as a JWT (RFC 7519) for interoperability with existing OAuth/OIDC infrastructure. When encoded as JWT:

- The intent_id MUST appear in the jti (JWT ID) claim
- The user_id MUST appear in the sub (subject) claim
- The org_id MUST appear in the iss (issuer) claim when present
- The not_before and not_after map directly to nbf and exp claims
- The full Intent Contract MUST appear in a custom intentid claim
- The signing algorithm MUST be EdDSA (Ed25519)

7.2 HTTP Header

When transmitting an AgentID over HTTP, implementations SHOULD use the following header:

```
X-IntentID-Agent: agent:acme_corp:john.doe%40acme.com:intentid:v1:a3f9...
X-IntentID-Contract: <base64url-encoded-intent-contract-jwt>
```

7.3 MCP Integration

For agents using the Model Context Protocol (MCP), the AgentID SHOULD be included in the MCP session initialization message as a metadata field. Tool call requests SHOULD include the AgentID to enable server-side verification gate enforcement.

8. Security Considerations

8.1 Key Management

The security of IntentID depends on the security of the private keys used to sign Intent Contracts. Implementations MUST follow key management best practices: keys MUST be generated with a cryptographically secure random number generator, private keys MUST be stored in a hardware security module (HSM) or equivalent secure enclave where possible, and keys MUST be rotatable without invalidating existing valid contracts.

8.2 Prompt Injection

IntentID provides a structural defense against prompt injection by making the system prompt a cryptographic anchor. However, implementations MUST also monitor for behavioral anomalies that could indicate a successful injection that exploits in-scope capabilities. The intent coherence check in the verification gate is the primary runtime defense.

8.3 Replay Attacks

Intent Contracts include not_before and not_after temporal bounds, providing replay protection. Implementations SHOULD additionally maintain a short-term nonce cache to prevent replay within the validity window. The not_after duration SHOULD be the minimum necessary for the intended task.

8.4 Registry Security

The IntentID registry — where AgentIDs are resolved to contracts — is a critical security component. It MUST be tamper-evident, highly available, and protected against unauthorized writes. Implementations SHOULD use a distributed, cryptographically authenticated registry.

9. Compliance Tiers

IntentID defines three compliance tiers to enable a practical adoption path without requiring full protocol compliance on day one. Each tier is a strict superset of the tier below it. Tier designation MUST be declared in the Intent Contract's goal_structure.compliance_tier field.

9.1 Individual Tier

Minimum viable IntentID for personal and developer use. Appropriate for single-user deployments, experimental agents, open-source projects, and development/staging environments. NOT recommended for production agents with access to sensitive data.

Requirement	Individual	Notes
AgentID construction (Section 3)	REQUIRED	Core identity primitive
Signed Intent Contract (Section 4)	REQUIRED	Ed25519 signature
goal_structure with forbidden_domains	REQUIRED	Explicit scope declaration
model_attestation (mode=api_hosted, null provider_attestation)	REQUIRED	Provider attestation not required
system_prompt_hash	REQUIRED	Anti-mutation anchor
tool_manifest with explicit actions	REQUIRED	No wildcards
sequence_rules	REQUIRED (empty array allowed)	Must be explicitly declared
Verification Gate (Sections 6.1)	REQUIRED	Steps 1-6 and 9-11
Audit log	RECOMMENDED	Append-only recommended
Contract Revocation (Section 3.5)	OPTIONAL	
Key rotation (Section 4.5)	OPTIONAL	
Intent coherence checking (Section 6.3)	NOT REQUIRED	
HSM key storage	NOT REQUIRED	

9.2 Professional Tier

For production deployments, team environments, SaaS products, and multi-agent workflows. Required when the agent has access to organizational data, communicates externally, or participates in a delegation chain. Recommended for any agent deployed beyond a single developer's personal use.

Requirement	Professional	Notes
All Individual tier requirements	REQUIRED	Superset of Individual
Contract Revocation — CRL (Section 3.5.1)	REQUIRED	CRL must be maintained
Contract Revocation — live endpoint (3.5.2)	RECOMMENDED	
Key rotation with kid versioning (Section 4.5)	REQUIRED	Overlap window pattern
Delegation chain validation (Section 5)	REQUIRED when delegated	
sequence_rules (at least one rule)	REQUIRED for read_write/execute scope	
Tamper-evident audit log	REQUIRED	Append-only with integrity
Verification Gate — all steps 1-11	REQUIRED	Including seq rules
Intent coherence checking (Section 6.3)	RECOMMENDED	
model_attestation — provider_attestation	RECOMMENDED	If provider supports it
HSM key storage	RECOMMENDED	

9.3 Enterprise Tier

For regulated industries, agents with access to financial, medical, legal, or privileged system resources, and any deployment subject to compliance frameworks (SOC 2, HIPAA, PCI DSS, FedRAMP). Enterprise tier provides the strongest security guarantees and is designed to satisfy regulatory audit requirements.

Requirement	Enterprise	Notes

All Professional tier requirements	REQUIRED	Superset of Professional
Intent coherence checking (Section 6.3)	REQUIRED	Reference taxonomy minimum
forbidden_domains populated (non-empty)	REQUIRED	Explicit exclusion list
sequence_rules (populated, non-empty)	REQUIRED	At least one rule per agent
model_attestation — valid provider_attestation	REQUIRED	Provider-signed attestation
HSM key storage for signing keys	REQUIRED	Or equivalent secure enclave
Tamper-evident audit log with crypto chaining	REQUIRED	Merkle-tree or equivalent
Live revocation endpoint (Section 3.5.2)	REQUIRED	
Custom taxonomy (Section 6.3.2)	REQUIRED for novel domains	
not_after duration <= 24 hours	REQUIRED for execute scope	Minimize blast radius
Delegation depth <= 3	REQUIRED	Unless explicitly justified

Tier Enforcement

Implementations MUST validate that the declared compliance_tier in the goal_structure matches the actual fields present in the contract. An agent claiming Enterprise tier but missing required Enterprise fields MUST be rejected by a conformant verification gate. This prevents tier downgrade attacks.

10. References

- RFC 2119 — Key words for use in RFCs to Indicate Requirement Levels
- RFC 7519 — JSON Web Token (JWT)
- RFC 8037 — CFRG Elliptic Curves for JOSE (Ed25519)
- RFC 8785 — JSON Canonicalization Scheme (JCS)
- RFC 8693 — OAuth 2.0 Token Exchange
- NIST SP 800-207 — Zero Trust Architecture
- NIST SP 800-63-4 — Digital Identity Guidelines
- NIST NCCoE Concept Paper — Accelerating the Adoption of Software and AI Agent Identity and Authorization (February 2026)
- OpenID Connect Core 1.0
- SPIFFE/SPIRE — Secure Production Identity Framework for Everyone
- MITRE ATT&CK for Enterprise — adversary tactic taxonomy (informative reference for forbidden_domains design)

Appendix A: IntentID Reference Taxonomy v1.0

This appendix defines the normative reference taxonomy for intent coherence checking. All IntentID implementations MUST support this taxonomy at minimum. The taxonomy is versioned and will evolve through the community process at github.com/cogumi/intentid-spec.

A.1 Agent Domain Definitions

Domain ID	Description	Primary Tool Categories	Adjacent Domains
software_development	Code creation, review, testing, debugging	code_editor, vcs, ci_cd, debugger, test_runner, package_manager	data_engineering, it_operations
customer_support	Handling customer inquiries and tickets	ticket_system, crm, knowledge_base, chat, email	content_creation, hr
finance	Financial operations, reporting, payments	accounting, payment, banking, erp, reporting	legal, data_engineering
legal	Contract and compliance management	contract_mgmt, compliance, document_review, legal_research	finance, hr
hr	Human resources and people operations	hris, payroll, recruiting, scheduling, performance_mgmt	legal, finance
it_operations	Infrastructure and system management	monitoring, deployment, cloud_mgmt, access_mgmt, ticketing	software_development, security
data_engineering	Data pipelines and warehousing	database, etl, data_warehouse, ml_pipeline, notebook	software_development, research

security	Security monitoring and response	siem, scanner, firewall, identity_mgmt, threat_intel	it_operations
content_creation	Creating and publishing content	cms, editor, media, publishing, seo	customer_support, research
research	Information gathering and analysis	search, document_store, citation_mgr, notebook, web_scraper	data_engineering, content_creation

A.2 Cross-Domain Distance Matrix

Distance values range from 0.0 (same domain) to 1.0 (maximally distinct). Values not listed default to 0.7. The default coherence threshold is 0.6 — tool calls with distance > 0.6 from the declared domain trigger escalation.

Domain Pair	Distance	Rationale
software_development ↔ data_engineering	0.2	Shared tooling and technical overlap
software_development ↔ it_operations	0.3	Deployment and DevOps adjacency
software_development ↔ security	0.4	Security tooling in dev pipelines
software_development ↔ research	0.5	Technical research is common
software_development ↔ finance	0.8	Distinct domains — low overlap
software_development ↔ hr	0.9	Distinct domains — minimal overlap
customer_support ↔ crm/ticket tools	0.1	Core tooling for this domain

customer_support ↔ content_creation	0.4	Knowledge base creation overlap
customer_support ↔ finance	0.7	Payment handling edge case
finance ↔ legal	0.3	Compliance and contract adjacency
finance ↔ hr	0.4	Payroll overlap
it_operations ↔ security	0.3	Access management overlap
data_engineering ↔ research	0.3	Analytical workflow overlap

A.3 Tool Category to Domain Mapping

Each tool registered in an IntentID tool manifest MUST declare a tool_category drawn from this taxonomy. The verification gate uses this category for coherence distance computation.

```
Tool categories and their primary domain assignments:

code_editor, vcs, ci_cd, debugger, test_runner  → software_development
ticket_system, crm, knowledge_base, chat         → customer_support
accounting, payment, banking, erp                → finance
contract_mgmt, compliance, document_review       → legal
hris, payroll, recruiting                        → hr
monitoring, deployment, cloud_mgmt               → it_operations
access_mgmt, identity_mgmt                       → security (also it_operations)
database, etl, data_warehouse, ml_pipeline       → data_engineering
siem, scanner, firewall, threat_intel            → security
cms, editor, media, publishing                   → content_creation
search, document_store, web_scraper              → research
email                                            → communication (cross-domain)
filesystem                                       → cross-domain (inherits from
agent domain)
web_browser                                      → cross-domain (inherits from
agent domain)
```

Cross-Domain Tools
Tools marked 'cross-domain' (email, filesystem, web_browser) are inherently multi-purpose. For these tools, coherence checking focuses on the action and data_scope rather than the tool category itself. A coding agent using filesystem:read on source_code paths is coherent. The same agent using filesystem:read on /hr/payroll/ paths is not.

Appendix B: Threat Model Summary

This appendix summarizes the IntentID threat model. The full STRIDE + ATT&CK hybrid analysis — including complete threat descriptions, attack vectors, and residual risk detail — is published as a companion document: IntentID Threat Model v1.0 at github.com/cogumi/intentid-spec/threat-model.

B.1 Adversary Classes Analyzed

- A1 — External Attacker: No authenticated access. Attacks from outside the trust boundary.
- A2 — Malicious Insider: Holds valid credentials. Motivated by sabotage or data exfiltration.
- A3 — Compromised Agent: The AI agent is manipulated while holding valid credentials.
- A4 — Supply Chain Attacker: Compromises model weights, tool providers, or registry.
- A5 — Prompt Injection Attacker: Embeds adversarial instructions in data the agent processes.

B.2 Protection Level Counts

- FULL protection (cryptographic prevention): 8 of 20 threats
- PARTIAL protection (reduces attack surface, limits blast radius): 10 of 20 threats
- OUT OF SCOPE (documented transparency): 2 of 20 threats

B.3 Key Fully-Protected Threats

- AgentID Forgery (T-S1): Ed25519 signature + registry verification makes forgery computationally infeasible
- Contract Modification (T-T1): IntentID hash + signature — any change is cryptographically detected
- System Prompt Substitution (T-T2): system_prompt_hash verified at instantiation
- Authorization Repudiation (T-R1): Ed25519 signature is non-repudiable
- Rate Limit Exhaustion (T-D2): Gate enforces rate limits on every tool call
- Delegation Chain Escalation (T-E1 + T-E4): child permissions ⊆ parent permissions, always enforced

B.4 Key Partial Threats and Primary Residual Risk

- Contract Replay (T-S2): Short not_after + live CRL required; nonce field planned for v0.3
- Key Compromise (T-S3): HSM required at Enterprise tier; anomaly detection recommended
- Multi-Step Exfiltration (T-I3): Sequence rules required; DLP at tool layer recommended
- Prompt Injection (T-E3): Gate constrains scope; model-level hardening is the primary defense
- Intent Coherence / Scope Creep (T-E2): REQUIRED at Enterprise tier; RECOMMENDED at Professional

B.5 Explicitly Out of Scope

- Physical key exfiltration from HSM hardware
- Model output misuse within declared scope (alignment/safety — model layer responsibility)
- Zero-day cryptographic breaks (post-quantum migration planned for v1.0)
- Privileged insider database attacks against registry infrastructure
- Social engineering of the human principal

Full threat model: github.com/cogumi/intentid-spec/threat-model

---

Feedback and contributions: github.com/cogumi/intentid-spec | intentid.org | vivek@cogumi.ai