COGUMI
Research & Innovation

IntentID

The Missing Trust Layer for the Agentic Internet

A Protocol Specification Proposal

Version 0.2 — February 2026
Vivek Chakravarthy Durairaj — Founder & CEO, Cogumi, Inc.
www.cogumi.ai

Executive Summary

The internet has a trust layer for connectivity (TCP/IP), a trust layer for encryption (TLS), and a trust layer for human authorization (OAuth). What it does not have is a trust layer for autonomous AI agents — and the absence of this layer is rapidly becoming the most consequential security gap of the AI era.

Today, every major enterprise is deploying AI agents with access to sensitive systems, customer data, financial infrastructure, and critical operations. These agents execute actions autonomously, spawn other agents, and operate across organizational boundaries — all without any standardized mechanism to verify who authorized them, what they are permitted to do, or whether their behavior remains consistent with their declared purpose.

Existing solutions address part of the problem. CyberArk and Palo Alto Networks extend Privileged Access Management to agents. Microsoft Entra Agent ID provides a directory for Azure-native agents. OAuth and OIDC provide token-based authentication. But none of these address the fundamental gap:

> **The Core Gap**
>
> *An AI agent's identity cannot be separated from its declared intent. An agent whose system prompt changes is no longer the same agent — yet no current identity standard captures this. Credentials can be stolen, behavioral signatures can be mimicked, but cryptographically anchored intent cannot be faked.*

IntentID is a proposed open protocol that solves this problem by making declared purpose a first-class cryptographic primitive in agent identity. Under IntentID, an agent's identity is not just who it is — it is what it is authorized to do, on whose behalf, with which tools, and within what constraints. Any change to the agent's purpose invalidates its identity, forcing re-authorization.

This whitepaper defines the IntentID protocol specification, explains the architectural gap it fills, positions it against existing standards, and proposes a path to open standardization. It is

submitted in response to the NIST NCCoE concept paper on AI Agent Identity and Authorization (February 5, 2026) and is open for community feedback.

1. The Problem: Agents Have No Identity

1.1 The Agentic Inflection Point

AI agents are no longer a future concept — they are in production. According to CyberArk research, 40% of enterprise financial institutions and software companies already have agentic AI in production, and 76% of organizations will have agents in production within three years. Salesforce Agentforce, Microsoft Copilot Studio, AWS Bedrock Agents, and hundreds of custom-built agent frameworks are being deployed across industries that manage some of the world's most sensitive data.

These agents do not just answer questions. They read and write to databases, send emails and messages on behalf of users, execute financial transactions, modify code in production systems, access customer records, and spawn other agents to complete sub-tasks. They act with the autonomy of a human employee but at the speed and scale of software.

1.2 Why Existing Identity Frameworks Fail

Every current identity framework was designed for a world without autonomous agents. The failures are structural, not fixable by configuration:

Identity Framework	What It Was Built For	Critical Gap for Agents
OAuth 2.0 / OIDC	Human user authorization	Session-based; no continuous intent verification
SAML	Enterprise SSO for humans	No concept of delegated agent authority
CyberArk PAM	Privileged human & machine accounts	Extends credentials, not intent; behavioral, not structural
Microsoft Entra Agent ID	Azure-native agent directory	Platform-locked; no cross-cloud intent anchoring
SPIFFE / SPIRE	Workload identity for microservices	Static workloads; no dynamic purpose binding

API Keys / Secrets	Service account access	Non-contextual; no delegation chain; no intent

The root cause of all these gaps is the same: existing systems authenticate the agent as an entity, but have no mechanism to verify that the agent's current behavior aligns with its original authorization. They answer 'who is this?' but cannot answer 'is this agent still doing what it was authorized to do?'

1.3 The Agent Mutability Problem

The deepest challenge — and the one no existing framework has solved — is that agents are fundamentally mutable. Unlike a human employee whose identity is stable, an AI agent's behavior, purpose, and risk profile can change completely with a single modification:

- A system prompt change transforms the agent's goals, tone, constraints, and capabilities
- A model version update alters reasoning patterns, knowledge, and behavioral tendencies
- A tool addition expands the agent's action surface without triggering re-authorization
- A prompt injection attack can silently redirect the agent's actions mid-session

This means behavioral monitoring — the approach taken by most current solutions — is insufficient. An attacker who understands a trusted agent's behavioral signature can craft a prompt injection that mimics that signature while exfiltrating data. The identity layer must be anchored to the agent's declared purpose at instantiation, not inferred from its behavior at runtime.

> **Key Insight**
> *Behavioral identity is a post-hoc observation. Cryptographic intent is a pre-commitment. Only the latter can be used as a security primitive.*

1.4 The Delegation Chain Problem

Modern agentic systems are not single-agent architectures. Orchestrator agents spawn sub-agents, which spawn further specialized agents, creating delegation chains of arbitrary depth. In multi-agent architectures, there is currently no standard mechanism to:

- Trace an action back to the originating human principal through multiple agent hops
- Enforce that delegated permissions cannot exceed the delegating agent's own authorization
- Cryptographically prove that a downstream agent is operating within the scope authorized by the chain above it
- Revoke an entire delegation chain when the root authorization is withdrawn

Without a delegation chain standard, the confused deputy problem — where a legitimate agent is manipulated into performing unauthorized actions — becomes impossible to prevent at scale.

2. The IntentID Protocol

IntentID is an open protocol specification that makes declared purpose a cryptographic primitive in agent identity. Its core thesis is that an agent's identity cannot be separated from its authorization context — and that any change to that context constitutes a new identity requiring fresh authorization.

2.1 The Identity Primitive

Every agent identity under IntentID is a composite of three components:

```
AgentID = OrgID + UserID + IntentID

Where:
  OrgID  := Organizational scope identifier (optional for individuals)
  UserID      := Verified human principal — always present; accountability anchor
  IntentID    := Cryptographic hash of the Intent Contract

Example:
  agent:acme_corp:john.doe@acme.com:sha256:a3f9c2b1e8d7...
```

The IntentID component is the key innovation. It is not a static identifier assigned to the agent — it is a deterministic hash of the agent's complete Intent Contract. Any change to the agent's declared purpose, tool manifest, model version, or system prompt produces a different IntentID, and therefore a different AgentID, which requires re-authorization before the agent can act.

2.2 The Intent Contract

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 signed by the issuing UserID (and OrgID where applicable) at agent instantiation.

```
{
  "intent_contract": {

    // Identity anchors
    "org_id":          "org:acme_corp",
    "user_id":             "usr:john.doe@acme.com",
    "parent_agent_id":     null,

    // Purpose declaration
    "declared_purpose":    "Process and respond to customer support tickets",
    "goal_structure": {
      "type":              "task_completion",
      "domain":            "customer_support",
      "scope":             "read_write",
      "targets":           ["tickets", "customer_records"],
      "forbidden_domains": ["finance", "hr", "legal"],
      "max_delegation_depth": 2,
      "compliance_tier":   "professional"
    },

    // Model attestation — any change breaks the IntentID
    "model_attestation": {
      "mode":             "api_hosted",
      "model_id":         "claude-sonnet-4-6",
      "provider":         "anthropic",
      "provider_attestation": { "attestation_id": "att_abc123", ... }
    },
    "system_prompt_hash":  "sha256:system_prompt_hash",

    // Tool manifest — explicit declaration of permitted capabilities
    "tool_manifest": [
      {
        "tool_id":          "zendesk_api",
        "tool_category":     "ticket_system",
        "allowed_actions":  ["read_ticket", "update_ticket", "close_ticket"],
        "data_scope":       "tickets.assigned_to_queue:customer_support",
        "rate_limit":       { "calls_per_minute": 60, "calls_per_day": 5000 }
      },
      {
        "tool_id":          "email_api",
        "tool_category":     "email",
        "allowed_actions":  ["send"],
        "data_scope":       "recipients:customer_domain_only",
        "rate_limit":       { "calls_per_hour": 100 }
      }
    ],

    // Action sequence constraints
    "sequence_rules": [
      {
```

```
        "rule_id":     "no-ticket-then-external-email",
        "pattern":     ["zendesk_api:read_ticket", "email_api:send"],
        "window":      5,
        "on_match":    "escalate",
        "unless":      "email.recipient in
contract.output_restrictions.allowed_recipients"
      }
    ],

    // Scope constraints
    "data_classification":  ["customer_pii", "support_tickets"],
    "output_restrictions":  { "no_external_domains": true, "no_attachments": true
},
    "escalation_triggers":  ["financial_data", "legal_matter", "media_inquiry"],

    // Temporal bounds
    "not_before":           "2026-02-22T00:00:00Z",
    "not_after":            "2026-03-22T23:59:59Z",

    // Cryptographic fields
    "issued_at":            "2026-02-22T09:15:00Z",
    "kid":                  "key_v2_abc123",
    "signature":            "<ed25519_signature_of_contract_by_user_id>",
    "intent_id":            "intentid:v1:<sha256_of_canonical_contract>"
  }
}
```

2.3 The Delegation Chain

When an agent spawns a child agent, the parent agent issues a delegated Intent Contract that must satisfy the following constraints:

- The child's OrgID and UserID must match the parent's
- The child's tool manifest must be a strict subset of the parent's — permissions can only narrow, never widen
- The child's parent_agent_id field must reference the parent's AgentID, creating a verifiable chain
- The delegation depth is explicitly declared and enforced — an agent cannot spawn agents beyond its authorized delegation depth

```
Human Principal (UserID: john.doe@acme.com)
  └── Orchestrator Agent (IntentID_A: full customer support scope)
        ├── Ticket Reader Agent (IntentID_B: read_ticket only)
        └── Response Writer Agent (IntentID_C: update_ticket + send email)
```

```
              └── Translation Agent (IntentID_D: text_only, no data access)

Rule: IntentID_D scope ⊆ IntentID_C scope ⊆ IntentID_A scope
Any violation of this constraint is rejected at the verification layer.
```

The delegation chain creates a cryptographically verifiable audit trail from every agent action back to the originating human principal, regardless of how many agent hops occurred.

2.4 Zero Trust Tool Access

Every tool invocation by an agent is validated in real time against the agent's Intent Contract. The v0.2 verification gate applies eleven checks in sequence, adding contract revocation, intent coherence, and action sequence constraint validation:

```
VERIFICATION GATE v0.2 — executed on every tool call:

1.  Contract validity   → Verify IntentID integrity + signature + revocation
status
2.  Tool authorization  → Is this tool in the agent's manifest?
3.  Action authorization → Is this specific action permitted for this tool?
4.  Data scope          → Does the requested data fall within declared scope?
5.  Output restriction  → Is the output destination permitted?
6.  Rate compliance     → Is the agent within declared rate limits?
7.  Intent coherence    → Is this action's semantic domain consistent with
                          the agent's declared goal_structure? (Enterprise tier)
8.  Sequence constraints → Does this action, in context of recent actions,
                          violate any declared sequence rules?
9.  Escalation triggers → Does this action match a human escalation condition?
10. Delegation validity → If delegated, does the chain remain unbroken?
11. Audit log           → Record decision with full context and kid reference

ALL PASS  → Action executes, logged immutably to audit ledger
ANY FAIL  → Action blocked, alert generated, optional human escalation
```

Step 7 — intent coherence — is what separates IntentID from credential-based systems. A coding agent authorized to use the filesystem attempting to access /hr/payroll/ paths will pass tool authorization (filesystem is in its manifest) but fail intent coherence, because file paths in the hr domain are semantically distant from the software_development domain declared in its goal_structure. The IntentID Reference Taxonomy defines the domain distance matrix that makes this check deterministic and interoperable.

Step 8 — sequence constraints — prevents multi-step confused deputy attacks where each individual action is in-scope but the sequence collectively constitutes unauthorized exfiltration or privilege escalation. A rule like 'database:read followed by email:send_external requires escalation' catches this pattern that step 2-5 would individually approve.

3. How IntentID Differs from Existing Approaches

IntentID is not a replacement for OAuth, OIDC, PAM, or existing identity infrastructure. It is a new layer — the intent layer — that sits above and complements existing systems. The distinction is fundamental:

Dimension	Existing Approaches	IntentID
Identity anchor	Credentials (keys, tokens, certs)	Declared intent + credentials
Mutation handling	Same credentials = same identity	Any change = new identity, new authorization
Delegation model	Inherited permissions (confused deputy risk)	Cryptographic chain, permissions narrow only
Prompt injection defense	Post-hoc behavioral monitoring	Pre-commitment; deviation breaks identity
Multi-agent trust	No standard exists	Native delegation chain with hash linkage
Platform scope	Vendor-specific (Azure, AWS, PANW)	Open, model-agnostic, cloud-agnostic
Authorization granularity	Resource-level (can access this API)	Action-level + intent-level (can do this action toward this goal)

3.1 IntentID vs. Microsoft Entra Agent ID

Microsoft Entra Agent ID provides a directory of agent identities within Azure tenants — essentially a registry that gives agents an account they can use to authenticate. It is a valuable foundation but has two structural limitations:

- It is Azure-native and Azure-locked. Enterprises with multi-cloud or hybrid environments cannot use it as a universal agent identity layer.
- It manages agent credentials, not agent intent. An Entra Agent ID does not change when the agent's system prompt changes — the same identity persists regardless of what the agent is now instructed to do.

IntentID is designed to be interoperable with Entra Agent ID — the UserID component can reference an Entra identity — while providing the intent layer that Entra does not.

3.2 IntentID vs. CyberArk Secure AI Agents

CyberArk's approach applies Privileged Access Management principles to AI agents — least privilege, just-in-time access, and session monitoring. This is the right instinct but architecturally limited by its PAM heritage:

- PAM was designed for stable machine identities (service accounts, certificates). Agents are fundamentally dynamic — their purpose changes, their tools change, their behavior changes.
- CyberArk monitors what agents do after the fact. IntentID establishes what agents are authorized to do before they act, making the authorization context a first-class cryptographic object.
- CyberArk has no standard for multi-agent delegation chains — a critical gap as orchestrator patterns become standard.

3.3 IntentID vs. OAuth 2.0 / OIDC Extensions

Several working groups are exploring extensions to OAuth for AI agents. These approaches have merit but face a structural challenge: OAuth was designed for discrete authorization events (a user grants an application access at login). Agent authorization is continuous, context-dependent, and action-level rather than session-level.

IntentID can be implemented as an extension to OAuth — the Intent Contract can be embedded in a JWT — but it represents a semantic upgrade: authorization is no longer about what resource the agent can access, but about what actions it can take toward what declared goals.

4. Alignment with NIST NCCoE Initiative

On February 5, 2026, NIST released a concept paper titled 'Accelerating the Adoption of Software and Artificial Intelligence Agent Identity and Authorization,' seeking public input to shape demonstration projects and future standards. The questions NIST posed directly map to the problems IntentID is designed to solve.

NIST Question	IntentID Response
Should agent identities be ephemeral or task-specific?	IntentID makes identities task-specific by design — each unique combination of purpose + tools + constraints produces a unique IntentID
How can zero-trust principles apply to agent authorization?	IntentID implements zero-trust at the action level — every tool call is verified against the intent contract in real time
How can agents prove authority for actions?	The delegation chain provides cryptographic proof from the originating human principal through every agent hop
How can organizations ensure agents log intent in tamper-proof ways?	The Intent Contract is cryptographically signed and immutable; the audit ledger records every action against the declared intent
Can authorization policies be dynamically updated when agent context changes?	Yes — context change = new IntentID = new authorization required, enforced structurally

Cogumi formally submits this whitepaper as a response to the NIST NCCoE concept paper and invites NIST to consider IntentID as a candidate framework for the demonstration project. We are prepared to implement a reference implementation in NCCoE labs and collaborate on the resulting practice guide.

5. Open Protocol Strategy

IntentID is proposed as an open protocol, not a proprietary product. The history of internet infrastructure demonstrates that open standards win at the protocol layer: TCP/IP, TLS, OAuth, JWT, and Kubernetes all succeeded because no single vendor owned them. IntentID follows this model deliberately.

5.1 Licensing and Governance

The IntentID specification will be published under the Apache 2.0 license — freely usable, modifiable, and implementable by any party, including commercial vendors. Reference implementations in Python, TypeScript, and Go will be open-sourced on the same terms.

For long-term governance, Cogumi proposes submitting IntentID to the OpenID Foundation — the body that governs OpenID Connect and related identity standards — as a working group specification. This ensures the protocol evolves through multi-stakeholder consensus rather than single-vendor control.

5.2 Interoperability Commitments

IntentID is designed to complement, not replace, existing infrastructure:

- UserID components can reference existing identity providers (Entra ID, Okta, Ping)
- Intent Contracts can be encoded as extensions to existing JWT structures
- The verification protocol is compatible with existing MCP, A2A, and agent framework architectures
- Existing PAM and IAM systems can be extended to enforce IntentID constraints without full replacement

5.3 Standards Roadmap

Cogumi proposes the following path to formal standardization:

Phase	Timeline	Milestone
0.1 — Specification	Q1 2026	Public whitepaper and draft specification published
0.2 — Reference Impl	Q2 2026	Open-source SDKs in Python, TypeScript, Go
0.3 — Framework Integration	Q2-Q3 2026	Native integrations with LangChain, CrewAI, AutoGen, Semantic Kernel
1.0 — OpenID Submission	Q3 2026	Formal submission to OpenID Foundation working group
1.1 — NIST Demonstration	Q4 2026	Reference implementation in NCCoE lab environment
2.0 — Ratified Standard	2027	Formal RFC or OpenID standard ratification

6. Ecosystem Implications

6.1 For Security Leaders (CISOs)

IntentID addresses the governance gap that security leaders identify as their top concern with agentic AI: accountability. Today, when an AI agent takes a damaging action, it is extremely difficult to determine who authorized it, whether the authorization was within policy, and whether the action was consistent with the agent's declared scope.

With IntentID, every action taken by every agent in an enterprise is traceable to a specific human principal, a specific declared purpose, and a specific set of constraints — all cryptographically anchored and immutably logged. Compliance reporting, incident response, and regulatory audits become tractable.

Critically, IntentID enables enterprises to confidently deploy agents in regulated environments — finance, healthcare, government — where current solutions leave too large an accountability gap. The EU AI Act, SEC cybersecurity disclosure rules, and emerging AI governance frameworks all demand exactly the kind of auditable, purposeful authorization that IntentID provides.

6.2 For Developers Building Agents

IntentID provides a simple, standardized way to declare what an agent is and what it can do — and to have those declarations enforced by the infrastructure rather than managed ad hoc in application code. A developer implementing IntentID can:

- Register an agent with a signed Intent Contract in under 10 lines of code
- Get automatic enforcement of tool scope and data access boundaries without writing custom authorization logic
- Receive instant feedback when an agent attempts to take an action outside its declared intent
- Generate compliance-ready audit trails automatically, without additional instrumentation

6.3 For AI Platform Providers

Agent platform providers — Salesforce, Microsoft, AWS, Google, and others — face increasing customer pressure around agent governance. IntentID provides a vendor-neutral standard that platforms can implement to give their customers confidence in agent security, without requiring customers to choose a single vendor's governance solution.

Platforms that implement IntentID gain a differentiating trust signal. Enterprises evaluating agent platforms will increasingly ask: does this platform support IntentID? The same dynamic drove OAuth adoption — platforms that didn't support it were excluded from enterprise consideration.

7. Call to Action

The window for establishing the IntentID standard is open now. In 12-24 months, major vendors will have proprietary agent authorization solutions entrenched in their platforms. The opportunity to establish an open, neutral standard that all vendors implement — rather than yet another vendor-locked identity silo — closes as incumbents deepen their moats.

Cogumi invites the following forms of engagement:

Design Partners

We are seeking 5-10 enterprise organizations currently deploying agentic AI who want to co-develop and validate the IntentID specification against real deployment challenges. Design partners will have direct input into the specification, early access to reference implementations, and co-authorship opportunities on the practice guide.

Framework Integrators

We are seeking engineers from LangChain, CrewAI, AutoGen, Microsoft Semantic Kernel, AWS Bedrock, and other agent frameworks to collaborate on native IntentID integrations. The goal is zero-friction adoption — developers should be able to add IntentID to any agent in under 10 lines of code.

Standards Contributors

We are forming a technical working group to refine the IntentID specification prior to formal OpenID Foundation submission. We welcome contributors with backgrounds in cryptographic protocol design, identity standards, AI security, and enterprise architecture.

NIST NCCoE Response

This whitepaper constitutes Cogumi's formal response to the NIST NCCoE concept paper on AI Agent Identity and Authorization. We request consideration of IntentID as a candidate

framework for the NCCoE demonstration project and are prepared to provide technical resources for implementation in NCCoE lab environments.

Get Involved

To engage with the IntentID working group, submit feedback on this specification, or inquire about design partnerships: contact@cogumi.ai | www.cogumi.ai/intentid

8. Conclusion

The agentic internet is being built right now, without a trust layer. Billions of autonomous AI actions will be taken in the coming years on behalf of individuals and organizations — reading sensitive data, executing transactions, communicating with customers, modifying critical systems — and the infrastructure to govern these actions does not yet exist in standardized form.

The internet learned, painfully, what happens when you build connectivity (TCP/IP) without encryption (TLS) — decades of insecurity that took years and enormous industry effort to retrofit. We are at the equivalent moment for agentic AI: the capabilities are being deployed faster than the trust infrastructure is being built.

IntentID is designed to be that trust infrastructure — an open, simple, cryptographically sound protocol that makes agent purpose a first-class identity primitive. The concept is straightforward: an agent is not just what it is, it is what it is authorized to do. Changing that authorization changes the identity. This single insight, implemented as an open standard, resolves the agent mutability problem, the delegation chain problem, and the intent accountability problem simultaneously.

We do not believe this problem will go unsolved. The question is whether it is solved by an open standard that the entire ecosystem benefits from, or by a collection of proprietary solutions that fragment enterprise agent governance and recreate the vendor-lock-in problems the cloud era spent a decade solving.

Cogumi's commitment is to the open standard. We invite the industry to build it with us.

About the Author

Vivek Chakravarthy Durairaj is the Founder & CEO of Cogumi, Inc. He brings extensive product management and technology leadership experience from Palo Alto Networks, where he led $1B+ security portfolios including work with App-ID, User-ID, and Content-ID — the identity primitives that directly inspired the IntentID architecture. He also held senior roles at Cisco and holds an MBA from UC Berkeley Haas. Contact: vivek@cogumi.ai

About Cogumi

Cogumi is an AI infrastructure company developing novel architectures for the next generation of AI systems. Our research spans AI model efficiency, agentic system design, and AI security protocols. The IntentID proposal emerges from our work on the fundamental requirements for trustworthy agentic AI deployment at enterprise scale.

www.cogumi.ai | contact@cogumi.ai

Appendix A: Intent Contract Schema Reference

The following represents the normative schema for an IntentID Intent Contract. All fields marked [REQUIRED] must be present for a valid Intent Contract. Fields marked [CONDITIONAL] are required under specific circumstances.

Field	Type	Description
org_id	string [CONDITIONAL]	Organizational identifier. Required for enterprise deployments.
user_id	string [REQUIRED]	Verified human principal. Must reference a verifiable identity.
parent_agent_id	string [CONDITIONAL]	Required when agent is spawned by another agent.
declared_purpose	string [REQUIRED]	Natural language description of the agent's authorized purpose.
goal_structure	object [REQUIRED]	Structured scope including domain, forbidden_domains, targets, compliance_tier, and max_delegation_depth. Anchor for intent coherence checking.
model_attestation	object [REQUIRED]	Model identity and integrity. Two modes: self_hosted (weight hash) and api_hosted (provider-signed attestation). Provider attestation required at Enterprise tier.
system_prompt_hash	string [REQUIRED]	SHA-256 hash of system prompt. Any change invalidates the IntentID.
tool_manifest	array [REQUIRED]	Explicit declaration of all permitted tools, allowed actions, tool_category, and rate limits. No wildcards.
sequence_rules	array [REQUIRED]	Action sequence constraints. Declare forbidden multi-step patterns. Empty array must be explicitly set.

data_classification	array [REQUIRED]	Data sensitivity classes the agent is permitted to access.
output_restrictions	object [REQUIRED]	Constraints on where the agent may send outputs.
escalation_triggers	array [REQUIRED]	Conditions that require human review before action proceeds.
not_before	datetime [REQUIRED]	Contract validity start time (ISO 8601).
not_after	datetime [REQUIRED]	Contract expiry time. For execute scope at Enterprise tier: max 24 hours.
kid	string [REQUIRED]	Key identifier. References the signing key version in the key registry.
signature	string [REQUIRED]	Ed25519 signature of the canonicalized contract by the UserID's private key.
intent_id	string [REQUIRED]	intentid:v1: + SHA-256 of canonical contract. The immutable agent identity.

Appendix B: Formal Specification — IntentID Construction

This appendix provides the formal pseudocode and mathematical specification for the IntentID construction. This constitutes the technical disclosure required for prior art purposes and for implementation by a person skilled in the art of cryptographic identity systems.

B.1 Intent Contract Canonicalization

Before hashing, the Intent Contract must be canonicalized to ensure deterministic output regardless of JSON key ordering or whitespace:

```
function canonicalize(contract):
  // 1. Remove signature field before hashing
  c = deep_copy(contract)
  delete c.signature
  delete c.intent_id
  // 2. Sort all object keys recursively (lexicographic)
  c = recursive_sort_keys(c)
  // 3. Serialize with no whitespace
  return json_serialize(c, whitespace=false)
```

B.2 IntentID Hash Construction

```
function compute_intent_id(contract):
  canonical = canonicalize(contract)
  return 'intentid:v1:' + hex(sha256(utf8_encode(canonical)))

// Example output:
// intentid:v1:a3f9c2b1e8d74f2c9a1b3e5d7f8c2a4b6e9d1f3a5c7b9e2d4f6a8c1b3e5d7f9
```

B.3 AgentID Construction

```
function construct_agent_id(org_id, user_id, intent_contract):
  intent_id = compute_intent_id(intent_contract)
  components = []
  if org_id != null:
    components.append(url_encode(org_id))
  components.append(url_encode(user_id))
  components.append(intent_id)
  return 'agent:' + ':'.join(components)

// With org:    agent:acme_corp:john.doe%40acme.com:intentid:v1:a3f9...
// Without org: agent:john.doe%40acme.com:intentid:v1:a3f9...
```

B.4 Intent Contract Signing

```
function sign_intent_contract(contract, private_key):
  canonical = canonicalize(contract)
  signature = ed25519_sign(private_key, utf8_encode(canonical))
  contract.signature = base64url_encode(signature)
  contract.intent_id = compute_intent_id(contract)
  return contract

function verify_intent_contract(contract, public_key):
  claimed_intent_id = contract.intent_id
```

```
  claimed_signature = contract.signature
  // Recompute intent_id
  computed_intent_id = compute_intent_id(contract)  // excludes sig + intent_id
  if computed_intent_id != claimed_intent_id:
    return INVALID  // contract was tampered
  // Verify signature
  canonical = canonicalize(contract)
  if not ed25519_verify(public_key, canonical,
base64url_decode(claimed_signature)):
    return INVALID  // signature mismatch
  return VALID
```

B.5 Delegation Chain Validation

```
function validate_delegation_chain(child_contract, parent_contract):
  // Rule 1: Principal must match
  assert child_contract.user_id == parent_contract.user_id
  if parent_contract.org_id != null:
    assert child_contract.org_id == parent_contract.org_id

  // Rule 2: Child tool manifest must be subset of parent
  parent_tools = {t.tool_id: t for t in parent_contract.tool_manifest}
  for child_tool in child_contract.tool_manifest:
    assert child_tool.tool_id in parent_tools
    parent_tool = parent_tools[child_tool.tool_id]
    // Actions must be subset
    assert set(child_tool.allowed_actions) <= set(parent_tool.allowed_actions)
    // Rate limits must be equal or lower
    assert child_tool.rate_limit <= parent_tool.rate_limit

  // Rule 3: Temporal bounds must be within parent bounds
  assert child_contract.not_before >= parent_contract.not_before
  assert child_contract.not_after <= parent_contract.not_after

  // Rule 4: Parent reference must be correct
  assert child_contract.parent_agent_id == construct_agent_id(
    parent_contract.org_id,
    parent_contract.user_id,
    parent_contract
  )
  return VALID
```

B.6 Verification Gate Algorithm

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

```
  // Step 1: Resolve and verify contract
  contract = resolve_contract(agent_id)  // lookup from registry
  if verify_intent_contract(contract, lookup_public_key(contract.user_id)) !=
VALID:
    return DENY(reason='invalid_contract')

  // Step 2: Temporal validity
  if timestamp < contract.not_before or timestamp > contract.not_after:
    return DENY(reason='temporal_bounds')

  // Step 3: Tool authorization
  tool_entry = find(contract.tool_manifest, tool_id=tool_id)
  if tool_entry is null:
    return DENY(reason='tool_not_in_manifest')

  // Step 4: Action authorization
  if action not in tool_entry.allowed_actions:
    return DENY(reason='action_not_permitted')

  // Step 5: Data scope
  if not data_ref.matches_scope(tool_entry.data_scope):
    return DENY(reason='data_out_of_scope')

  // Step 6: Output restriction
  if not output_dest.satisfies(contract.output_restrictions):
    return DENY(reason='output_restricted')

  // Step 7: Rate compliance
  if rate_exceeded(agent_id, tool_id, tool_entry.rate_limit):
    return DENY(reason='rate_limit')

  // Step 8: Escalation check
  for trigger in contract.escalation_triggers:
    if trigger.matches(action, data_ref):
      return ESCALATE(reason=trigger, notify=contract.user_id)

  // Step 9: Delegation chain (if delegated agent)
  if contract.parent_agent_id != null:
    parent = resolve_contract(contract.parent_agent_id)
    if validate_delegation_chain(contract, parent) != VALID:
      return DENY(reason='delegation_chain_invalid')

  // All checks passed
  audit_log(agent_id, tool_id, action, data_ref, output_dest, timestamp)
  return ALLOW
```

Appendix C: Glossary

Term	Definition
AgentID	The complete composite identity: OrgID + UserID + IntentID
IntentID	The SHA-256 hash of an agent's Intent Contract; changes with any modification to the contract
Intent Contract	Cryptographically signed machine-readable declaration of agent purpose, tools, scope, and constraints
Delegation Chain	The cryptographically linked sequence of AgentIDs from the originating human principal through all spawned agents
Intent Coherence	The property of an agent's action sequence being consistent with its declared purpose
Confused Deputy	A security vulnerability where a legitimate agent is manipulated into performing actions outside its authorization
Verification Gate	The real-time authorization check applied to every tool invocation by an agent
Audit Ledger	The immutable, tamper-evident log of all agent actions with their associated Intent Contracts