IntentID Test Vectors

Version 1.0 — Cryptographically Verified

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

IntentID Spec Reference: OpenSpec v0.2

Keypair: RFC 8032 §6 Test Vectors (seed-derived)

Reference Implementation: intentid_reference.py (Apache 2.0)

Machine-Readable Vectors: intentid_test_vectors.json

Repository: github.com/dkeviv/intentID

How to Use This Document

All values in this document are computed by the IntentID reference implementation (intentid_reference.py) using the RFC 8032 §6 test keypair. A conformant IntentID implementation MUST produce byte-for-byte identical outputs for the inputs specified in each vector. The machine-readable JSON file (intentid_test_vectors.json) contains the same vectors in a format suitable for loading directly into automated test harnesses. Both files are published at github.com/cogumi/intentid-spec/test-vectors.

1. Test Keypair

All signing and verification test vectors use the RFC 8032 §6 keypair. These keys are published solely for test purposes. They MUST NOT be used in any production deployment.

```
// RFC 8032 §6 Test Keypair — FOR TESTING ONLY
// Source: RFC 8032 Section 6 (seed-derived via Ed25519PrivateKey.from_private_bytes)

PRIVATE KEY SEED (hex, 32 bytes):
  9d61b19deffd5a60ba844af492ec2cc44449c5697b326919703bac031cae3d55

PUBLIC KEY (hex, 32 bytes):
  700e2ce7c4b674427eab27ba820bcf6f0faebe68e09fe8564292114e41dc6a41

// Alternate keypair (used for 'wrong key' verification test VER-5):
ALT PUBLIC KEY (hex, 32 bytes):
  c61e278621027598ce2ee4cea835ec4a485b781fa89b97ab754fb7676d319ac2

// WARNING: NEVER use these keys in production.
// They are published here and in the test vectors JSON.
```

How to load in Python

from cryptography.hazmat.primitives.asymmetric.ed25519 import Ed25519PrivateKeyprivate_key = Ed25519PrivateKey.from_private_bytes(bytes.fromhex("9d61b19deffd5a60ba844af492ec2cc44449c5697b326919703bac031cae3d55"))

2. JSON Canonicalization (RFC 8785 JCS)

JCS canonicalization is the foundation of IntentID's tamper detection. Two implementations that produce different canonical forms for the same input will compute different IntentIDs. These 12 vectors cover all edge cases defined in RFC 8785.

ID	Description	Input (JSON)	Expected Canonical Output
JCS-1	Key ordering	{"z":1,"a":2,"m":3}	{"a":2,"m":3,"z":1}
JCS-2	Nested key sort	{"outer":{"z":1,"a":2},"alpha":true}	{"alpha":true,"outer":{"a":2,"z":1}}
JCS-3	Array order preserved	{"items":[3,1,2],"name":"test"}	{"items":[3,1,2],"name":"test"}
JCS-4	Null value	{"x":null,"y":"hello"}	{"x":null,"y":"hello"}
JCS-5	String escaping — \n \t	{"s":"hello\nworld\t!"}	{"s":"hello\nworld\t!"}
JCS-6	Backslash and quote	{"s":"say \"hello\" \\world"}	{"s":"say \"hello\" \\world"}
JCS-7	Integer vs float	{"i":42,"f":3.14}	{"f":3.14,"i":42}
JCS-8	Boolean values	{"t":true,"f":false}	{"f":false,"t":true}
JCS-9	Empty object and array	{"empty_obj":{},"empty_arr":[]}	{"empty_arr":[],"empty_obj":{}}
JCS-10	Deeply nested mixed types	{"a":{"b":{"c":[1,null,true,"x"]}}}	{"a":{"b":{"c":[1,null,true,"x"]}}}
JCS-11	Unicode passthrough	{"name":"Vivek Chakravarthy Durairaj"}	{"name":"Vivek Chakravarthy Durairaj"}
JCS-12	Control char escaping	{"s":"\u0000\u001f"}	{"s":"\u0000\u001f"}

3. IntentID Hash Construction

The IntentID is computed as: 'intentid:v1:' + hex(sha256(utf8(jcs_canonicalize(contract_without_sig_and_intent_id)))). These vectors verify that JCS + SHA-256 produce the expected values, and that any contract modification produces a different IntentID.

3.1 System Prompt Hash (embedded in base contract)

The base contract uses the following system prompt hash. All IID-1 vectors are derived from a contract containing this value:

```
system_prompt_text:  "You are a customer support agent for Acme Corp."
sha256(utf8(above)): 9476e4eea7399250a9c4d0d8434685a3de54d8c2eedd4e1c1f2e38ee066452bf
```

3.2 Hash Vectors

ID	Description	Expected IntentID	Notes
IID-1	Base contract — reference vector	intentid:v1:c74b86f0c6decc74091a2395c223c6bd2801cd...	All implementations MUST match exactly
IID-2	declared_purpose modified post-construction	intentid:v1:6faa7f6fc476731f942363513e9cc2780bbccd...	IID-2 ≠ IID-1 — tamper detection
IID-3	system_prompt_hash changed	intentid:v1:a51449c5b989cef257495cdc4d8782d3473e46...	IID-3 ≠ IID-1 — prompt substitution detection
IID-4	Minimal contract (null org_id, empty arrays)	intentid:v1:18eb36f3c51c178682fdc10bd9639868b72153...	Null and empty field handling

Full IntentID values (exact match required):

```
IID-1: intentid:v1:c74b86f0c6decc74091a2395c223c6bd2801cdfa66facb5f556c65c6796f90e6
IID-2: intentid:v1:6faa7f6fc476731f942363513e9cc2780bbccde539d0019051c4002c8d5272f4
IID-3: intentid:v1:a51449c5b989cef257495cdc4d8782d3473e4691d15c2fd3fa1fe9bd82c67f9c
IID-4: intentid:v1:18eb36f3c51c178682fdc10bd9639868b72153d62f40c36c461fa906a2c43d41
```

4. AgentID Construction

AgentID = 'agent:' [org_id ':'] user_id ':' intent_id, where org_id and user_id are URL-encoded. These vectors verify correct URL encoding behavior.

ID	Description	Inputs	Expected AgentID (truncated)	Encoding Notes
AID-1	With OrgID (standard enterprise)	org=acme_corp, user=john.doe@acme.com	agent:acme_corp:john.doe%40acme.com:intentid:v1:c74b...	@ → %40
AID-2	Without OrgID (individual)	org=null, user=alice@example.com	agent:alice%40example.com:intentid:v1:18eb...	No org segment
AID-3	Spaces and + in identifiers	org='org with spaces', user=user+tag@example.com	agent:org%20with%20spaces:user%2Btag%40example.com:...	space→%20, +→%2B, @→%40

Full AgentID values (exact match required):

```
AID-1:
agent:acme_corp:john.doe%40acme.com:intentid:v1:c74b86f0c6decc74091a2395c223c6bd2801cdfa66facb5f556c65c6796f90e6
AID-2:
agent:alice%40example.com:intentid:v1:18eb36f3c51c178682fdc10bd9639868b72153d62f40c36c461fa906a2c43d41
AID-3:
agent:org%20with%20spaces:user%2Btag%40example.com:intentid:v1:18eb36f3c51c178682fdc10bd9639868b72153d62f40c36c461fa906a2c43d41
```

5. Intent Contract Signing

Ed25519 is a deterministic signature scheme. Given the same private key, message, and canonicalization, the signature MUST be identical across all conformant implementations. These vectors verify determinism.

REQUIRED: Determinism

Ed25519 signing is deterministic (RFC 8032 §5.1). If your implementation produces a different signature for SIGN-1 than the value below, your canonicalization or signing implementation is non-conformant. The two most common failure modes are: (1) non-deterministic JSON key ordering before hashing, (2) incorrect JCS implementation that differs for specific field types.

```
// SIGN-1: Standard contract signing
// Input: base_contract_unsigned + kid='rfc8032_tv1' + issued_at='2026-01-15T09:00:00Z'
// Private key: RFC 8032 §6 seed (see Section 1)

Expected intent_id:
intentid:v1:3fe52514da758e2c37d514c354545a863978cdbf1bcd9664208b021635020321

Expected signature:
0V_zfzGXhYnb0CWoWjVwisxNr-mE_ji4cYS6ND5VdOl0F3RJjp_mEbg1OoMJEAeeQa-240mWlfgcEWcLHJ23Bw

// SIGN-2: Determinism check
// Same inputs as SIGN-1 → MUST produce identical intent_id and signature
// Expected to equal SIGN-1 exactly.
```

6. Intent Contract Verification

Verification checks five conditions in order: (1) IntentID hash integrity, (2) signature validity, (3) revocation status, (4) temporal bounds. These vectors cover the valid case and one failure per condition.

ID	Description	Condition Tested	Expected Result	Expected Reason Prefix
VER-1	Valid contract — all checks pass	All conditions met	**VALID**	valid
VER-2	Tampered content — field modified post-signing	IntentID mismatch	**INVALID**	intent_id_mismatch
VER-3	Expired contract — not_after='2026-02-01T00:00:00Z', eval at 2026-03-01	Temporal: expired	**INVALID**	expired
VER-4	Not yet valid — not_before='2026-06-01T00:00:00Z', eval at 2026-03-01	Temporal: future	**INVALID**	not_yet_valid
VER-5	Wrong public key — signed with key1, verified with key2	Signature mismatch	**INVALID**	invalid_signature

Evaluation timestamp for VER-3 and VER-4

All verification vectors use a fixed evaluation time of 2026-03-01T12:00:00Z. Implementations must support passing a fixed 'now' for deterministic testing.

7. Delegation Chain Validation

These vectors cover all five delegation rules from OpenSpec Section 5. Each failure vector tests exactly one rule violation. Implementations must enforce all rules.

ID	Description	Rule Tested	Expected Result	Expected Reason Prefix
DEL-1	Valid delegation — child is proper subset of parent	All rules pass	**VALID**	valid
DEL-2	Child has delete_ticket — not in parent manifest	Rule 2: scope narrowing (actions)	**INVALID**	actions_exceed_parent
DEL-3	Child has payroll_system — tool not in parent	Rule 2: scope narrowing (tools)	**INVALID**	tool_not_in_parent
DEL-4	Child not_after='2027-12-31' — exceeds parent '2026-12-31'	Rule 3: temporal containment	**INVALID**	child_not_after_exceeds_parent
DEL-5	Child user_id='attacker@evil.com' — parent='john.doe@acme.com'	Rule 1: principal preservation	**INVALID**	principal_mismatch
DEL-6	Child rate=120/min — exceeds parent rate=60/min	Rule 2: scope narrowing (rate limits)	**INVALID**	rate_limit_exceeds_parent

8. Verification Gate — All 11 Steps

These vectors test each gate step in isolation. GATE-ALLOW verifies the baseline (all checks pass). Each subsequent vector introduces exactly one failure, confirming the gate catches it at the correct step. Step 10 (delegation chain) cross-references the delegation vectors.

ID	Gate Step	Failure Condition	Expected Result	Expected Reason Prefix
GATE-ALLOW	ALL (baseline)	No failure — all checks pass	ALLOW	all_checks_passed
GATE-S1a	Step 1 — contract integrity	declared_purpose modified after signing	DENY	contract_invalid:intent_id_mismatch
GATE-S1b	Step 1 — revocation	intent_id in CRL	DENY	contract_revoked
GATE-S1c	Step 1 — temporal	not_after='2026-02-01', eval at 2026-03-01	DENY	contract_invalid:expired
GATE-S2	Step 2 — tool authorization	tool_id='payroll_system' not in manifest	DENY	tool_not_in_manifest
GATE-S3	Step 3 — action authorization	action='delete_ticket' not in allowed_actions	DENY	action_not_permitted
GATE-S4	Step 4 — data scope	data_scope='hr/payroll/march2026', allowed='tickets/'	DENY	data_out_of_scope
GATE-S5	Step 5 — output restriction	output_dest='external:attacker.com', no_external=true	DENY	output_restricted
GATE-S6	Step 6 — rate compliance	calls_per_minute exceeded (_test_exceeded=true)	DENY	rate_limit_exceeded
GATE-S7	Step 7 — intent coherence	payroll tool for customer_support agent (dist=0.7>0.6)	ESCALATE	intent_coherence_anomaly
GATE-S8	Step 8 — sequence rules	session=[zendesk:read_ticket], then email:send matches rule	ESCALATE	sequence_rule_triggered
GATE-S9	Step 9 — escalation triggers	data_scope contains 'legal_matter' pattern	ESCALATE	escalation_trigger
GATE-S10	Step 10 — delegation chain	See DEL-2 through DEL-6 (cross-reference)	DENY	delegation_*

Gate Step 7 — Intent Coherence Parameters

GATE-S7 uses: agent domain = 'customer_support', tool_category = 'payroll', tool domain = 'hr'. Domain distance('customer_support', 'hr') = 0.7. Default threshold = 0.6. Since 0.7 > 0.6, the gate returns ESCALATE.

Gate Step 8 — Sequence Rule Parameters

GATE-S8 uses: session_window = ['zendesk:read_ticket'], candidate = 'email:send'. Rule pattern = ['zendesk:read_ticket', 'email:send'], window = 5, on_match = 'escalate'. The pattern appears as a subsequence in (recent + candidate), so the gate returns ESCALATE.

9. Action Sequence Rule Evaluation

These vectors test the subsequence matching logic and window boundary behavior. The reference rule is: pattern=['zendesk:read_ticket','email:send'], window=5, on_match='escalate'.

ID	Description	session_window (before candidate)	Candidate Action	Expected Result
SEQ-1	No prior actions — pattern cannot match	[]	email:send	CONTINUE
SEQ-2	Immediately preceding — pattern matches	["zendesk:read_ticket"]	email:send	ESCALATE
SEQ-3	read_ticket beyond window — not in last 4	["a:b","c:d","e:f","g:h","i:j"]	email:send	CONTINUE
SEQ-4	Non-adjacent subsequence — read_ticket then other then send	["zendesk:update_ticket","zendesk:read_ticket","zendesk:close_ticket"]	email:send	ESCALATE

SEQ-3 Window Boundary Explanation

Window=5 means: consider the last (window-1)=4 items from session_window PLUS the candidate. Session window has 5 items; only the last 4 are considered. read_ticket is at position 0 (oldest), which falls outside the window. Therefore the pattern does not match.

SEQ-4 Subsequence Matching Explanation

The subsequence check finds pattern items in order but not necessarily adjacent. ['update_ticket', 'read_ticket', 'close_ticket'] + 'send' contains both 'read_ticket' (at index 1) and 'send' (at index 3) in order, so the pattern ['read_ticket','send'] matches as a subsequence.

10. Intent Coherence Checking

These vectors test the domain distance calculation against the IntentID Reference Taxonomy v1.0. The default coherence threshold is 0.6. Actions with distance > 0.6 from the declared domain trigger ESCALATE. Actions in a declared forbidden_domain trigger ESCALATE with distance=1.0 regardless of the matrix value.

ID	Agent Domain	Tool Category	Tool Domain	Distance	Expected Result
COH-1	customer_support	ticket_system	customer_support	0.0	COHERENT
COH-2	customer_support	email	cross_domain	0.0	COHERENT
COH-3	customer_support	payroll	hr	0.7	ANOMALY
COH-4	customer_support	banking	finance	1.0	ANOMALY
COH-5	software_development	database	data_engineering	0.2	COHERENT
COH-6	software_development	cms	content_creation	0.6	COHERENT
COH-7	software_development	crm	customer_support	0.7	ANOMALY

COH-4: Forbidden Domain Override

COH-4 uses tool_category='banking' which maps to domain='finance'. The goal_structure declares forbidden_domains=['finance','hr','legal']. Since 'finance' is in forbidden_domains, the distance is set to 1.0 and result is ANOMALY regardless of the matrix distance value.

COH-6: Threshold Boundary — distance exactly equals threshold

COH-6 has distance=0.6 and threshold=0.6. The check is: distance > threshold (strict greater-than). Since 0.6 is NOT > 0.6, the result is COHERENT. Implementations MUST use strict greater-than, not greater-than-or-equal.

11. Conformance Checklist

A conformant IntentID implementation MUST pass all vectors in categories 2 through 10. The following checklist summarizes the minimum passing criteria for each compliance tier.

Vector Category	# Vectors	Individual	Professional	Enterprise
JCS Canonicalization (all 12)	12	REQUIRED	REQUIRED	REQUIRED
IntentID Hash (IID-1 to IID-4)	4	REQUIRED	REQUIRED	REQUIRED
AgentID Construction (AID-1 to AID-3)	3	REQUIRED	REQUIRED	REQUIRED
Signing determinism (SIGN-1, SIGN-2)	2	REQUIRED	REQUIRED	REQUIRED
Contract verification (VER-1 to VER-5)	5	REQUIRED	REQUIRED	REQUIRED
Delegation chain (DEL-1 to DEL-6)	6	RECOMMENDED	REQUIRED	REQUIRED
Gate ALLOW + Steps 1-6 (GATE-S1a through GATE-S6)	8	REQUIRED	REQUIRED	REQUIRED
Gate Step 7 — Intent coherence (GATE-S7)	1	NOT REQUIRED	RECOMMENDED	REQUIRED
Gate Step 8 — Sequence rules (GATE-S8)	1	NOT REQUIRED	REQUIRED	REQUIRED
Gate Step 9 — Escalation triggers (GATE-S9)	1	REQUIRED	REQUIRED	REQUIRED
Gate Step 10 — Delegation (GATE-S10, cross-ref DEL)	1	RECOMMENDED	REQUIRED	REQUIRED
Sequence rule edge cases (SEQ-1 to SEQ-4)	4	NOT REQUIRED	REQUIRED	REQUIRED
Intent coherence (COH-1 to COH-7)	7	NOT REQUIRED	RECOMMENDED	REQUIRED

12. Running the Reference Implementation

The reference implementation generates and verifies all test vectors programmatically. Every value in this document is the direct output of that implementation.

```
# Install dependency
pip install cryptography

# Generate all test vectors (runs all assertions, writes JSON)
python3 intentid_reference.py

# Expected output:
# IntentID Reference Implementation v1.0
# Generating test vectors...
# Generated 56 test vectors across 9 categories:
#   tv_jcs: 12 vectors
#   tv_intentid: 4 vectors
#   tv_agentid: 3 vectors
#   tv_signing: 2 vectors
#   tv_verification: 5 vectors
#   tv_delegation: 6 vectors
#   tv_gate: 13 vectors
#   tv_sequence: 4 vectors
#   tv_coherence: 7 vectors
# Written: intentid_test_vectors.json
# All assertions PASSED — vectors are cryptographically verified.
```

To test your implementation against the vectors, load intentid_test_vectors.json and verify your outputs match the expected_* fields for each vector. Any mismatch indicates a non-conformant implementation.

---

github.com/cogumi/intentid-spec | intentid.org | vivek@cogumi.ai