MolTrust Protocol Technical Specification v0.5

The MolTrust Protocol: Technical Specification

Version 0.5

MolTrust / CryptoKRI GmbH, Zurich March 2026

This document is a companion to *The MolTrust Protocol: A Verification Standard for Autonomous Software Agents* (Whitepaper v0.6.1). It provides the technical definitions, data models, verification flows, and conformance requirements referenced in that document.

Architectural Overview

This specification is organized around three distinct layers. Readers and implementers should be clear about which layer they are working with, as conformance requirements differ across layers.

Layer A — Protocol Standard The normative core. Defines data formats, signing rules, verification flows, and lifecycle semantics. Any independent implementation that conforms to Layer A can interoperate with any other conformant implementation at the evidence level.

Layer B — Reference Registry The MolTrust-operated service layer. Defines how the reference implementation exposes identity resolution, credential revocation, trust score queries, and on-chain anchoring. Conformance to Layer B is required to interoperate with the MolTrust reference API. Other operators MAY run conformant registries using different infrastructure.

Layer C — Reference Reputation Model An informative scoring model used by the MolTrust reference registry. Other implementations MAY use different scoring models provided they consume Layer A evidence formats. Score portability across implementations with different models is a format guarantee, not a semantic guarantee.

This distinction resolves the central design tension in any open-standard-plus-canonical-service architecture: the protocol is genuinely open at Layer A; the reference service has operator-specific policy at Layer B; the scoring model is reproducible but not mandatory at Layer C.

Table of Contents

1. Scope and Terminology
2. Data Model (Layer A)
    - 2.1 Signed Payload Boundary
    - 2.2 Agent DID Document

  - 2.3 Authorization Credential
  - 2.4 Interaction Proof
  - 2.5 Vertical Identifiers
  - 2.6 Endorsement
  - 2.7 Violation Record
  - 2.8 Agent Authorization Envelope (AAE) — Formal Schema
  - 2.9 Trust Tier 0 Credential Schema
  - 2.10 Sybil Resistance Mechanisms
  - 2.11 Credential TTL and Revocation Protocol
3. Verification Flow (Layer A)
  - 3.1 Identity Verification
  - 3.2 Pre-Transaction Verification Flow
  - 3.3 Authorization Verification
  - 3.4 Behavioral History Verification
  - 3.5 Interaction Proof Verification
4. Reference Reputation Model (Layer C — Informative)
5. Reference Registry (Layer B)
6. On-Chain Anchoring (Layer B)
7. Chain-Agnostic Anchoring (Layer B)
8. Credential Lifecycle (Layer A)
9. Agent Lifecycle (Layer A)
10. Threat Model
11. Privacy Model
12. Worked Example
13. Conformance

---

1. Scope and Terminology

1.1 Scope

This specification defines:

- The data formats for agent identity documents, verifiable credentials, interaction proofs, endorsements, and violation records (Layer A)
- The verification procedures for identity, authorization, and behavioral history (Layer A)
- The lifecycle rules for credentials and agents (Layer A)
- The reference trust scoring model (Layer C — informative)
- The reference registry API and policy (Layer B)
- On-chain anchoring formats and requirements (Layer B)
- A threat model for the verification system
- Privacy principles and data minimization requirements
- Conformance requirements differentiated by layer

This specification does not define:

- What agents are permitted to do
- How disputes between agents are adjudicated
- The legal validity of any credential or interaction proof
- Which blockchain network must be used for anchoring
- How agents are built or operated internally

1.2 Terminology

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

Agent — an autonomous software process that acts on behalf of a principal.

Principal — the human, organization, or other agent on whose behalf an agent acts. A principal MUST have a stable DID that persists across agent registrations.

DID — Decentralized Identifier, as specified in W3C DID Core v1.0.

DID Document — the document associated with a DID, containing public keys and service endpoints.

Verifiable Credential (VC) — a tamper-evident claim about a subject, as specified in W3C VC Data Model 2.0.

Interaction Proof — a signed artifact produced after an interaction between two agents, attesting to the fact and outcome of that interaction.

Trust Score — a numeric value in [0, 100] derived from an agent's behavioral record and endorsement graph, computed by a registry using a defined scoring model.

Endorsement — a signed attestation by one agent that another agent has behaved reliably in a defined vertical.

Vertical — a domain of agent activity, expressed as a namespaced string of the form `<namespace>/<identifier>` (see Section 2.5).

Seed Agent — an agent registered with a bootstrap weight by a registry operator, used to initialize the endorsement graph. Bootstrap weights are time-limited (see Section 8.2).

Verifier — any party that checks a DID, credential, interaction proof, or trust score.

Issuer — any party that signs and issues a Verifiable Credential.

On-chain anchor — a transaction on a distributed ledger that permanently records a hash of a protocol artifact.

Registry — a service that resolves DIDs, maintains revocation lists, computes trust scores, and records interaction proofs. The MolTrust reference registry is one such service.

Violation Record — a signed artifact attesting to a confirmed protocol violation by an identified agent (see Section 2.7).

Protocol conformance — conformance to Layer A requirements.

Registry conformance — conformance to Layer B requirements.

AAE (Agent Authorization Envelope) — a structured policy object embedded in or accompanying a Verifiable Credential that defines an agent's permitted actions, operational constraints, and validity parameters. Formally specified in Section 2.8.

Trust Tier 0 — the highest identity assurance level in MolTrust, backed by KYC verification from an accredited provider. See Section 2.9.

CAEP (Continuous Access Evaluation Protocol) — a framework for real-time revocation signaling between issuers and verifiers, referenced in Section 2.11.

Runtime Control Plane — the logical component responsible for evaluating AAE constraints at transaction time, including action matching, limit enforcement, and jurisdiction checks. See Section 3.2.

1.3 Notation

JSON examples in this document use `<placeholder>` notation for variable values. All JSON objects MUST be serialized using RFC 8785 canonical JSON before signing. Field ordering in examples is illustrative; canonical ordering is determined by RFC 8785.

---

2. Data Model (Layer A)

2.1 Signed Payload Boundary

For all signed artifacts in this protocol, the following rules apply:

1. The signed payload is the RFC 8785 canonical JSON serialization of the object, **excluding** the `proof` field (or `proofInitiator`/`proofResponder` fields for interaction proofs)
2. Additional fields not defined in this specification MAY be present in objects; they are included in the canonical serialization and therefore covered by the signature
3. Fields MUST NOT be added to a signed object after signing
4. The signing algorithm is Ed25519 as specified in Ed25519Signature2020
5. Signature values are encoded as base58btc multibase strings

This boundary applies to all MolTrust-defined signed artifacts: Authorization Credentials, Interaction Proofs, Endorsements, and Violation Records.

DID Documents are explicitly excluded from this signing boundary. DID Document integrity is guaranteed by the DID method's own resolution mechanism (e.g. registry lookup, blockchain anchoring, or HTTPS binding), not by an embedded proof block. Verifiers MUST authenticate DID Documents per the rules of the applicable DID method, not by checking for a `proof` field.

2.2 Agent DID Document

Each agent MUST have a DID conforming to W3C DID Core v1.0. Implementations MAY use any conformant DID method. The `did:moltrust` method is defined and operated by the MolTrust reference registry.

Mandatory fields:

```
{
  "@context": [
    "https://www.w3.org/ns/did/v1",
    "https://moltrust.ch/ns/v1"
  ],
  "id": "did:moltrust:<unique-identifier>",
  "verificationMethod": [
    {
      "id": "did:moltrust:<id>#keys-1",
      "type": "Ed25519VerificationKey2020",
      "controller": "did:moltrust:<id>",
      "publicKeyMultibase": "<base58btc-encoded-public-key>"
    }
  ],
  "authentication": ["did:moltrust:<id>#keys-1"],
  "assertionMethod": ["did:moltrust:<id>#keys-1"]
}
```

Optional fields:

Field	Type	Notes

`service`	array	Service endpoints including registry reference
`created`	ISO 8601	Timestamp of DID creation
`updated`	ISO 8601	Timestamp of last update
`controller`	DID string	Principal DID — SHOULD be present for sub-agents
`alsoKnownAs`	array	Cross-registry references e.g. ERC-8004 agent ID
`keyAgreement`	array	For encrypted communication channels

Key rotation: When rotating keys, the agent MUST add the new key to `verificationMethod` with a new key ID, update `authentication` and `assertionMethod` to reference the new key, and retain the old key entry marked with `"revoked": true` and a `revokedDate`. This preserves a verifiable timeline of key epochs.

2.3 Authorization Credential

An agent’s authority to act on behalf of a principal MUST be expressed as a W3C Verifiable Credential.

Mandatory fields:

```
{
  "@context": [
    "https://www.w3.org/2018/credentials/v1",
    "https://moltrust.ch/ns/credentials/v1"
  ],
  "type": ["VerifiableCredential", "AuthorizationCredential"],
  "id": "<uuid-v4>",
  "issuer": "did:<method>:<principal-id>",
  "issuanceDate": "2026-03-22T00:00:00Z",
  "expirationDate": "2027-03-22T00:00:00Z",
  "credentialSubject": {
    "id": "did:moltrust:<agent-id>",
    "authorizedBy": "did:<method>:<principal-id>",
    "permittedActions": ["<action-type>"],
    "vertical": "<namespace>/<identifier>",
    "constraints": {}
  },
  "proof": {
    "type": "Ed25519Signature2020",
    "created": "2026-03-22T00:00:00Z",
    "verificationMethod": "did:<method>:<principal-id>#keys-1",
    "proofPurpose": "assertionMethod",
    "proofValue": "<base58btc-signature>"
  }
}
```

`permittedActions` values:

Value	Meaning
`transact`	May execute financial transactions
`delegate`	May authorize sub-agents
`endorse`	May issue endorsements
`verify`	May request verification of other agents

`publish`	May publish content on behalf of principal
`*`	All actions permitted (use with caution)

Implementations MAY define additional action types using the namespace convention `<namespace>/<action>`.

Delegation chains: An agent MAY issue an AuthorizationCredential to a sub-agent. The sub-agent's `permittedActions` MUST be a subset of the delegating agent's own authorized actions. Verifiers MUST traverse the full chain from subject to root principal. Maximum delegation depth: 8 hops. Verifiers MUST reject chains exceeding this depth.

Constraints: The `constraints` field is a free-form JSON object for policy-specific restrictions (e.g. `{"maxTransactionValue": 5000}`). Constraint semantics are application-defined and not normatively specified in this document. For a formal, machine-evaluable constraint model, see the Agent Authorization Envelope (Section 2.8).

2.4 Interaction Proof

An Interaction Proof is the primary evidence artifact. It is produced after an interaction that both parties wish to record.

Mandatory fields:

```json
{
  "@context": "https://moltrust.ch/ns/interaction/v1",
  "type": "InteractionProof",
  "id": "<uuid-v4>",
  "session": "<session-id>",
  "initiator": {
    "did": "did:moltrust:<initiator-id>",
    "vertical": "<namespace>/<identifier>"
  },
  "responder": {
    "did": "did:moltrust:<responder-id>",
    "vertical": "<namespace>/<identifier>"
  },
  "timestamp": "2026-03-22T10:00:00Z",
  "outcome": "<outcome-value>",
  "outcomeHash": "sha256:<hex-hash>",
  "proofInitiator": {
    "type": "Ed25519Signature2020",
    "verificationMethod": "did:moltrust:<initiator-id>#keys-1",
    "proofValue": "<base58btc-signature>"
  },
  "proofResponder": {
    "type": "Ed25519Signature2020",
    "verificationMethod": "did:moltrust:<responder-id>#keys-1",
    "proofValue": "<base58btc-signature>"
  }
}
```

Field semantics:

Field	Semantics
`id`	Globally unique identifier (UUID v4). Duplicate IDs MUST be rejected by registries.
`session`	Application-defined session identifier. Multiple proofs MAY reference the same session (e.g. multi-step interactions).

	Session IDs are scoped to the initiating agent and are not globally unique.
`outcome`	One of: `completed`, `partial`, `disputed`, `failed`
`outcomeHash`	SHA-256 of the canonical outcome payload (see below)

Outcome hash construction: The hash input MUST be the RFC 8785 canonical JSON serialization of an outcome object containing at minimum:

```
{
  "proofId": "<same uuid-v4 as proof id>",
  "timestamp": "<same timestamp as proof>",
  "outcome": "<same outcome value>",
  "summary": "<free-form string, not exceeding 256 characters>"
}
```

The `summary` field MAY contain a human-readable description of the outcome. Raw transaction data, counterparty personal information, and financial amounts MUST NOT be included in the hashed payload. The full outcome object MAY be retained locally by the parties; only the hash is submitted to the registry.

Signing procedure (sequential — not parallel):

1. Initiator constructs the proof object with all mandatory fields, leaving `proofResponder` absent
2. Initiator computes the RFC 8785 canonical serialization of the object without `proofResponder`
3. Initiator signs the canonical bytes with Ed25519 and adds `proofInitiator`
4. Initiator transmits the partially-signed object to the responder
5. Responder verifies `proofInitiator` against the initiator's DID Document
6. Responder computes the RFC 8785 canonical serialization of the full object **including `proofInitiator`** but excluding `proofResponder`
7. Responder signs these canonical bytes — which now cover the initiator's signature — and adds `proofResponder`
8. Either party MAY submit the completed proof to a registry

Critical: The responder's signature in step 7 covers the initiator's signature from step 3. This is sequential, not parallel. Independent implementations MUST follow this exact order or the signatures will be incompatible. A parallel signing scheme (both parties signing the same "naked" payload) is not conformant.

One-sided proofs: If the responder is unavailable or refuses to sign within a reasonable timeout (application-defined, SHOULD be at least 300 seconds), the initiator MAY submit the proof with `"singleSig": true` and `proofResponder` absent. One-sided proofs are valid Layer A artifacts but carry reduced weight in Layer C scoring.

Multi-agent interactions: Interactions involving more than two agents SHOULD be modeled as multiple bilateral proofs sharing the same `session` identifier. There is no native multi-party proof format in v0.3.

2.5 Vertical Identifiers

Verticals are expressed as namespaced strings of the form `<namespace>/<identifier>`.

Format rules: - Namespace and identifier MUST contain only alphanumeric characters, hyphens, and underscores - Namespace and identifier MUST be separated by exactly one forward slash - Total length MUST NOT exceed 128 characters - Case is significant: `moltrust/Travel` and `moltrust/travel` are different verticals

The `moltrust/` namespace is reserved for verticals defined and published by MolTrust. Current defined values:

Value	Domain
`moltrust/travel`	Travel booking and logistics
`moltrust/commerce`	Agentic commerce and purchasing
`moltrust/prediction`	Prediction markets and forecasting
`moltrust/skill`	Skill verification and auditing
`moltrust/finance`	Financial transactions and DeFi
`moltrust/identity`	Identity verification services
`moltrust/general`	General-purpose, cross-domain

Any party MAY define verticals in their own namespace (e.g. `acme/logistics`, `custom/my-domain`). There is no registration requirement. The cross-vertical bonus in the reference scoring model (Section 4) treats verticals with different namespaces OR different identifiers within the same namespace as distinct for diversity calculation purposes.

2.6 Endorsement

```json
{
  "@context": "https://moltrust.ch/ns/endorsement/v1",
  "type": "SkillEndorsementCredential",
  "id": "<uuid-v4>",
  "issuer": "did:moltrust:<endorser-id>",
  "issuanceDate": "2026-03-22T00:00:00Z",
  "expirationDate": "2026-06-22T00:00:00Z",
  "credentialSubject": {
    "id": "did:moltrust:<endorsed-id>",
    "vertical": "<namespace>/<identifier>",
    "weight": 1.0,
    "basis": "interaction-proofs",
    "evidenceCount": 5,
    "evidenceSummaryHash": "<sha256-of-supporting-proof-ids>"
  },
  "proof": {
    "type": "Ed25519Signature2020",
    "verificationMethod": "did:moltrust:<endorser-id>#keys-1",
    "proofValue": "<base58btc-signature>"
  }
}
```

Endorsement rules:

- Maximum validity: 365 days. Default SHOULD be 90 days.
- An endorser MUST NOT issue more than one active endorsement per (endorsed DID, vertical) pair. A new endorsement in the same vertical supersedes the previous one.
- The `basis` field MUST be one of: `interaction-proofs` (endorsed based on observed interaction proofs), `delegation` (parent agent endorsing sub-agent), `operator` (registry operator bootstrap endorsement, time-limited).
- The `evidenceSummaryHash` is a SHA-256 hash of the canonical JSON array of interaction proof IDs supporting the endorsement. For `operator` basis endorsements, this field MAY be absent.
- Principals MAY issue endorsements. Seed agents MAY endorse

other agents but MUST NOT endorse each other using `operator` basis after the bootstrap period.
- The `weight` field (0.0–1.0) is endorser-declared and represents the endorser’s confidence level.

2.7 Violation Record

A Violation Record is a signed artifact attesting that a confirmed protocol violation has been recorded against an agent.

```
{
  "@context": "https://moltrust.ch/ns/violation/v1",
  "type": "ViolationRecord",
  "id": "<uuid-v4>",
  "issuanceDate": "2026-03-22T00:00:00Z",
  "subject": {
    "agentDid": "did:moltrust:<violating-agent-id>",
    "principalDid": "did:<method>:<principal-id>"
  },
  "violation": {
    "type": "<violation-type>",
    "interactionProofId": "<uuid-v4-of-disputed-proof>",
    "description": "<free-form string, max 512 chars>"
  },
  "adjudication": {
    "adjudicatorType": "external",
    "adjudicatorReference": "<URL, case ID, or contract address>",
    "confirmedAt": "2026-03-22T12:00:00Z"
  },
  "registrySignature": {
    "type": "Ed25519Signature2020",
    "verificationMethod": "did:moltrust:registry#keys-1",
    "proofValue": "<base58btc-signature>"
  }
}
```

Violation types:

Value	Meaning
`identity-spoofing`	Agent misrepresented its identity
`authorization-abuse`	Agent acted outside declared permissions
`sybil`	Agent participated in a confirmed sybil cluster
`behavioral-fraud`	Agent deliberately deceived a counterparty
`clone-impersonation`	Clone represented as original agent

Recording process:

1. A party submits a disputed interaction proof (outcome = `disputed`) to the registry
2. The registry records the dispute without adjudicating it
3. External adjudication confirms or rejects the dispute (legal, arbitral, or contractual)
4. On confirmation, the registry records a ViolationRecord signed by the registry operator key
5. The ViolationRecord is permanently associated with both `agentDid` and `principalDid`
6. If stake is held, forfeiture is executed by the registry smart contract

Appeal and reversal: If an external adjudicator reverses a decision, the registry MUST record a `ViolationReversal` object and mark the original ViolationRecord as `"reversed": true`. Reversed violation records MUST NOT contribute to score computation.

A ViolationReversal object MUST contain the following fields:

```
{
  "@context": "https://moltrust.ch/ns/violation/v1",
  "type": "ViolationReversal",
  "id": "<uuid-v4>",
  "reversedRecordId": "<id of original ViolationRecord>",
  "reversalDate": "2026-03-22T15:00:00Z",
  "adjudicatorReference": "<URL, case ID, or contract address>",
  "registrySignature": {
    "type": "Ed25519Signature2020",
    "verificationMethod": "did:moltrust:registry#keys-1",
    "proofValue": "<base58btc-signature>"
  }
}
```

The `registrySignature` is computed over the RFC 8785 canonical serialization of the object excluding the `registrySignature` field, using the same registry operator key as ViolationRecord. Registries MUST make ViolationReversal objects queryable by `reversedRecordId`.

2.8 Agent Authorization Envelope (AAE) — Formal Schema

The Agent Authorization Envelope (AAE) is a machine-evaluable policy object that replaces or extends the free-form `constraints` field in an AuthorizationCredential (Section 2.3). An AAE provides a deterministic, auditable authorization boundary for autonomous agent actions.

An AAE MAY be embedded directly in an AuthorizationCredential’s `credentialSubject` as the `authorizationEnvelope` field, or MAY be issued as a standalone Verifiable Credential of type `AgentAuthorizationEnvelope`.

The AAE consists of three top-level objects: `mandate`, `constraints`, and `validity`.

2.8.1 Mandate

The `mandate` object defines WHAT the agent is permitted to do.

```
{
  "mandate": {
    "purpose": ["commerce", "data_read"],
    "allowedActions": [
      "https://moltrust.ch/actions/transact",
      "https://moltrust.ch/actions/query/*"
    ],
    "deniedActions": [
      "https://moltrust.ch/actions/query/admin/*"
    ],
    "resources": [
      "https://api.example.com/bookings/*",
      "https://api.example.com/inventory/read"
    ],
    "delegation": {
      "allowed": false,
      "maxSubAgents": 0,
      "maxDepth": 0,
      "attenuationOnly": true
    }
  }
}
```

Field definitions:

Field	Type	Description
`purpose`	string[]	REQUIRED. Enumerated values: `commerce`, `data_read`, `data_write`, `communication`, `delegation`, `administration`. At least one value MUST be present.
`allowedActions`	string[]	REQUIRED. URI patterns defining permitted actions. Supports wildcard `*` for path segments. Default-deny: any action not matching an `allowedActions` pattern is denied.
`deniedActions`	string[]	OPTIONAL. URI patterns explicitly denied. Takes precedence over `allowedActions`. If an action matches both `allowedActions` and `deniedActions`, it is DENIED.
`resources`	string[]	OPTIONAL. URI patterns defining the ABAC object layer — which resources the agent may act upon. When present, both action AND resource must match for authorization.
`delegation`	object	OPTIONAL. Controls whether the agent may delegate authority to sub-agents.

Delegation sub-fields:

Field	Type	Default	Description
`allowed`	boolean	`false`	Whether delegation is permitted at all.
`maxSubAgents`	integer	`0`	Maximum number of sub-agents this agent may authorize.
`maxDepth`	integer	`0`	Maximum further delegation depth from this agent. MUST NOT exceed 8.
			If `true`,

`attenuationOnly`	boolean	`true`	delegated AAEs MUST be strictly equal to or more restrictive than the parent AAE. Sub-agents MUST NOT gain permissions the parent does not hold.

2.8.2 Constraints

The `constraints` object defines the operational boundaries WITHIN which permitted actions may be executed.

```json
{
  "constraints": {
    "duration": {
      "ttl": 86400,
      "maxSessionDuration": 3600,
      "allowedDays": [1, 2, 3, 4, 5],
      "allowedHours": {
        "start": 8,
        "end": 18
      },
      "timezone": "Europe/Zurich"
    },
    "limits": {
      "autonomousThreshold": 500.00,
      "stepUpThreshold": 2000.00,
      "approvalThreshold": 10000.00,
      "maxTransactionsPerHour": 20,
      "currency": "USDC"
    },
    "scope": {
      "jurisdictions": ["CH", "DE", "AT", "FR"],
      "counterpartyMinScore": 40
    },
    "obligations": {
      "requireHumanApprovalAbove": 5000.00,
      "toolAllowlist": [
        "https://moltrust.ch/tools/mt_shopping_verify",
        "https://moltrust.ch/tools/mt_travel_verify"
      ]
    }
  }
}
```

Duration sub-fields:

Field	Type	Description
`ttl`	integer	Time-to-live in seconds from `validity.issuedAt`. Maximum: 86400 (24 hours) for autonomous agents, 604800 (7 days) for supervised agents.
`maxSessionDuration`	integer	Maximum duration of a single session in seconds.

`allowedDays`	integer[]	Days of the week when the agent may operate. 1 = Monday through 7 = Sunday (ISO 8601 weekday numbering).
`allowedHours`	object	Time window within allowed days. `start` and `end` are integers 0–23 representing hours in the specified timezone.
`timezone`	string	IANA timezone identifier (e.g. `"Europe/Zurich"`, `"America/New_York"`). REQUIRED when `allowedDays` or `allowedHours` is present.

Limits sub-fields:

Field	Type	Description
`autonomousThreshold`	number	Maximum transaction amount the agent may execute without any additional checks.
`stepUpThreshold`	number	Transaction amount above which additional verification is required (e.g. re-authentication, additional credential presentation).
`approvalThreshold`	number	Transaction amount above which explicit human approval is required.
`maxTransactionsPerHour`	integer	Rate limit on transactions per rolling hour.
`currency`	enum	Currency for threshold values. One of: `USDC`, `EUR`, `CHF`, `USD`.

Scope sub-fields:

Field	Type	Description
`jurisdictions`	string[]	ISO 3166-1 alpha-2 country codes where the agent may operate. Empty array means unrestricted.
`counterpartyMinScore`	integer	Minimum trust score (0–100) required for any counterparty the agent interacts with.

Obligations sub-fields:

Field	Type	Description
`requireHumanApprovalAbove`	number	Transaction value above which human-in-the-loop approval is mandatory, regardless of other thresholds.
`toolAllowlist`	string[]	URI patterns of tools/APIs the agent is permitted to invoke. When present, tool invocations not matching any pattern MUST be denied.

2.8.3 Validity

The `validity` object defines WHO issued the envelope, to WHOM it is bound, and WHEN it is active.

```
    {
      "validity": {
        "issuer": "did:moltrust:<principal-id>",
        "holderBinding": "did:moltrust:<agent-id>",
        "issuedAt": "2026-03-25T00:00:00Z",
        "expiresAt": "2026-03-26T00:00:00Z",
        "revocationEndpoint":
"https://api.moltrust.ch/revocation/aae/<envelope-id>",
        "onChainAnchor": {
          "chain": "base-mainnet",
          "block": 43800000,
          "txHash": "0xabc123..."
        }
      }
    }
```

Field definitions:

Field	Type	Required	Description
`issuer`	DID string	REQUIRED	The DID of the principal or parent agent issuing this envelope.
`holderBinding`	DID string	REQUIRED	The DID of the agent to which this envelope is bound. The envelope MUST NOT be used by any other agent.
`issuedAt`	ISO 8601 datetime	REQUIRED	Timestamp of envelope creation.
`expiresAt`	ISO 8601 datetime	REQUIRED	Expiration timestamp. No open-ended

			credentials are permitted.
`revocationEndpoint`	URL	REQUIRED	HTTPS endpoint where verifiers can check if this envelope has been revoked.
`onChainAnchor`	object	OPTIONAL	On-chain reference for the envelope hash, providing tamper evidence.

On-chain anchor sub-fields:

Field	Type	Description
`chain`	string	Chain identifier (e.g. `"base-mainnet"`, `"ethereum-mainnet"`).
`block`	integer	Block number containing the anchor transaction.
`txHash`	string	Transaction hash of the anchor.

2.8.4 Validation Rules

The following rules MUST be enforced by any conformant AAE evaluator:

1. **Default-deny:** Any action not explicitly matched by an `allowedActions` pattern MUST be denied.
2. **Deny precedence:** If an action matches both `allowedActions` and `deniedActions`, the action MUST be denied. `deniedActions` always takes precedence.
3. **Delegation depth cap:** `delegation.maxDepth` MUST NOT exceed 8. AAEs specifying a value greater than 8 MUST be rejected.
4. **Mandatory expiry:** `validity.expiresAt` MUST be present and MUST be a future timestamp at evaluation time. AAEs without `expiresAt` MUST be rejected unconditionally.
5. **Holder binding:** The presenting agent's DID MUST match `validity.holderBinding`. Mismatched presentations MUST be rejected.
6. **Attenuation enforcement:** When `delegation.attenuationOnly` is `true`, any delegated AAE MUST have `allowedActions` that are a subset of the parent's effective allowed actions (after deny exclusion), `limits` that are equal to or more restrictive than the parent's, and `scope.jurisdictions` that are a subset of the parent's.
7. **TTL ceiling:** `constraints.duration.ttl` MUST NOT exceed 86400 seconds for autonomous agents or 604800 seconds for supervised agents. Evaluators MUST reject AAEs exceeding these ceilings.

2.9 Trust Tier 0 Credential Schema

Trust Tier 0 represents the highest identity assurance level in MolTrust: a developer or operator identity backed by KYC verification from an accredited provider. Tier 0 credentials bridge the pseudonymous DID layer to a verified real-world identity without exposing personal data on-chain.

JSON-LD Schema:

```
{
  "@context": [
    "https://www.w3.org/2018/credentials/v1",
    "https://moltrust.ch/ns/tier0/v1"
  ],
  "type": ["VerifiableCredential", "DeveloperIdentityCredential"],
  "id": "<uuid-v4>",
  "issuer": "<accredited-kyc-provider-did>",
  "issuanceDate": "<iso-8601>",
  "expirationDate": "<iso-8601>",
  "credentialSubject": {
    "id": "<developer-did>",
    "verificationLevel": "tier0_kyc",
    "kycProvider": "<provider-name>",
    "verifiedAt": "<iso-8601>",
    "validUntil": "<iso-8601>",
    "jurisdiction": "<ISO-3166-1>"
  },
  "credentialStatus": {
    "id": "https://api.moltrust.ch/revocation/<credential-id>",
    "type": "BitstringStatusListEntry",
    "statusListIndex": "<integer>",
    "statusListCredential":
"https://api.moltrust.ch/revocation/status-list/tier0"
  },
  "proof": {
    "type": "Ed25519Signature2020",
    "created": "<iso-8601>",
    "verificationMethod": "<kyc-provider-did>#keys-1",
    "proofPurpose": "assertionMethod",
    "proofValue": "<base58btc-signature>"
  }
}
```

Field semantics:

Field	Description
`issuer`	The DID of the accredited KYC provider. MUST be a provider recognized by the registry operator.
`verificationLevel`	Fixed value `"tier0_kyc"`. Indicates full KYC verification has been completed.
`kycProvider`	Human-readable name of the KYC provider (e.g. `"Sumsub"`, `"Onfido"`, `"IDnow"`).
`verifiedAt`	ISO 8601 timestamp of when the KYC verification was completed.
`validUntil`	ISO 8601 timestamp of when the KYC verification expires. MUST NOT exceed 365 days from `verifiedAt`.
`jurisdiction`	ISO 3166-1 alpha-2 code of the jurisdiction where the identity was verified.

Issuance rules:

- The KYC provider MUST be accredited by the registry operator. The list of accredited providers is published at `https://<registry-domain>/.well-known/kyc-providers.json`.

- The credential MUST NOT contain any personal data (name, address, date of birth, document numbers). The `credentialSubject` attests that verification occurred, not what was verified.
- Maximum validity: 365 days. Renewal requires re-verification.
- The credential MUST include a `credentialStatus` field referencing the revocation registry (see Section 2.11).

Trust Tier hierarchy:

Tier	Assurance Level	Backing
Tier 0	Highest	KYC-verified real-world identity
Tier 1	High	Organizational credential (e.g. domain-verified, ERC-8004)
Tier 2	Medium	Behavioral reputation only (interaction proofs, endorsements)
Tier 3	Low	Self-declared, no external verification

Verifiers MAY use trust tier as a gating criterion. For example, a financial services vertical MAY require Tier 0 for agents transacting above a defined threshold.

2.10 Sybil Resistance Mechanisms

Sybil attacks — where a single adversary creates multiple fake identities to manipulate trust scores, endorsement graphs, or governance — represent a fundamental threat to any decentralized reputation system. MolTrust employs a layered defense combining cryptographic proofs, economic costs, permanent records, and graph analysis.

2.10.1 Dual-Signature Interaction Proofs

Every bilateral interaction proof (Section 2.4) requires sequential signatures from both the initiator and the responder. The responder’s signature covers the initiator’s signature, creating a non-repudiable chain of commitment. This means:

- A single adversary cannot fabricate bilateral proofs without controlling two distinct signing keys.
- One-sided proofs (where the responder does not sign) are valid but carry reduced weight in scoring (Section 4.2), limiting their value for sybil manipulation.
- The sequential signing procedure prevents parallel forgery — the initiator’s commitment is locked before the responder signs.

2.10.2 Economic Stake via x402

The x402 payment protocol introduces a measurable economic cost for interactions with trusted endpoints. When agents must pay (in USDC on Base L2) to access scored endpoints, credential issuance, or premium verification services, the cost of creating and maintaining fake identities scales linearly:

- Each sybil identity must independently fund x402 payments.
- Credential issuance fees (e.g. $5.00 per VC) make mass sybil creation economically unfavorable.
- The payment record on Base L2 provides an independent audit trail of agent activity.

2.10.3 Violation Records

Violation Records (Section 2.7) serve as permanent, portable negative attestations:

- Confirmed sybil participation results in a `sybil` violation type, permanently associated with both the agent DID and the principal DID.
- Violation records are anchored on-chain (Section 6.1), making them tamper-proof and publicly auditable.
- Principal DID continuity (Section 8.5) ensures that re-registration under a new agent DID does not escape a principal's violation history.

2.10.4 Endorsement Decay (v0.3 Roadmap)

Endorsements that are not renewed through continued interaction lose their contribution to trust scores over time. The time decay function `d_i = exp(-0.005 * age_in_days)` (Section 4.2) ensures that:

- Inactive sybil clusters degrade naturally as endorsements age without renewal.
- Maintaining a high trust score requires ongoing, genuine interactions.
- An adversary must continuously invest resources to sustain sybil identities, rather than front-loading endorsements and abandoning the identity.

Future versions MAY introduce explicit endorsement expiry notifications and mandatory renewal windows.

2.10.5 Graph Anomaly Detection (v0.3 Roadmap)

Jaccard Cluster Detection identifies groups of agents with suspiciously overlapping endorser sets:

- For any two agents A and B, compute Jaccard similarity: `J(A,B) = |endorsers_A ∩ endorsers_B| / |endorsers_A ∪ endorsers_B|`
- Threshold: `J > 0.8` triggers a sybil investigation flag.
- Flagged clusters are subject to manual review by the registry operator.
- Vertical diversity is a secondary signal: agents operating in fewer than 3 distinct verticals while exhibiting high Jaccard similarity receive a penalty of 10.0 points in the scoring model.

Graph anomaly detection is a heuristic. It does not catch sophisticated adversaries who deliberately diversify endorser sets across identities. Registry operators running large networks SHOULD supplement Jaccard detection with spectral clustering, community detection (e.g. Louvain algorithm), or temporal pattern analysis.

2.11 Credential TTL and Revocation Protocol

All credentials issued under the MolTrust protocol MUST carry an explicit expiration timestamp. Open-ended credentials are not permitted.

2.11.1 Maximum TTL by Credential Type

Credential Type	Maximum TTL	Notes
AuthorizationCredential	365 days	Renewable; renewal generates new credential ID

SkillEndorsementCredential	90 days	Default SHOULD be 90 days; maximum 365 days
InteractionProof	72 hours	Validity window for submission to registry; the record is permanent once accepted
ViolationRecord	Permanent	No expiration. May be marked `reversed` but never deleted.
DeveloperIdentityCredential (Tier 0)	365 days	Requires re-verification for renewal
AgentAuthorizationEnvelope	86400 seconds (autonomous) / 604800 seconds (supervised)	See Section 2.8

2.11.2 Revocation Registry Endpoint

Every credential with a revocation requirement MUST reference a revocation endpoint in its `credentialStatus` field or in the AAE’s `validity.revocationEndpoint`.

Endpoint specification:

```
GET /revocation/{credential-id}
```

Response format:

```
{
  "credentialId": "<credential-id>",
  "revoked": false,
  "revokedAt": null,
  "reason": null,
  "checkedAt": "2026-03-25T10:00:00Z"
}
```

When revoked:

```
{
  "credentialId": "<credential-id>",
  "revoked": true,
  "revokedAt": "2026-03-25T09:30:00Z",
  "reason": "key_compromise",
  "checkedAt": "2026-03-25T10:00:00Z"
}
```

Revocation reason values:

Value	Meaning
`key_compromise`	The signing key has been compromised
`issuer_revocation`	The issuer has explicitly revoked the credential
`subject_request`	The credential subject requested revocation
`policy_violation`	The credential was revoked due to a policy violation

`superseded`	The credential has been replaced by a newer version
`expiry_acceleration`	The credential is being expired ahead of its natural TTL

2.11.3 Bitstring Status List Compatibility

The MolTrust revocation registry is compatible with the W3C Bitstring Status List v1.0 specification. Each credential type maintains a separate status list credential:

```
GET /revocation/status-list/{credential-type}
```

The status list is a compressed bitstring where each bit position corresponds to a `statusListIndex` assigned at credential issuance. A bit value of `1` indicates the credential at that index has been revoked. This allows verifiers to check revocation status with a single HTTP request for the entire status list, rather than individual queries per credential.

Status lists MUST be signed by the registry operator key and MUST include a `validUntil` timestamp (maximum 300 seconds from issuance) to prevent stale caching.

2.11.4 Verifier Behavior

Verifiers MUST implement the following revocation checking logic:

1. **Expired credential:** If `expirationDate` (or `expiresAt` for AAEs) is in the past at verification time (UTC), the credential MUST be denied. There is no grace period.
2. **Revoked credential:** If the revocation endpoint returns `"revoked": true`, the credential MUST be denied.
3. **Unreachable revocation endpoint:** If the revocation endpoint is unreachable (network error, timeout, HTTP 5xx), the verifier SHOULD deny the credential (fail-closed). Verifiers with explicit risk tolerance policies MAY accept credentials with unreachable revocation endpoints, but MUST log the event and re-check within 60 seconds.
4. **Cache policy:** Verifiers MAY cache revocation responses for up to 300 seconds. Cached responses beyond this window MUST trigger a fresh check.

This fail-closed default is deliberate. In an autonomous agent economy, the cost of accepting a revoked credential (unauthorized action, financial loss) typically exceeds the cost of rejecting a valid credential (temporary service disruption, retry).

---

3. Verification Flow (Layer A)

3.1 Identity Verification

1. **Resolve** the agent’s DID to its DID Document
2. **Validate** that the DID Document is well-formed per Section 2.2
3. **Check** that the DID is not revoked by consulting a valid revocation source. Layer A conformant implementations MUST support revocation checking; the revocation source itself is implementation-defined. Layer B conformant registries provide the reference revocation endpoint (Section 5.3). Verifiers operating without a Layer B registry MUST implement an alternative revocation mechanism (e.g., local revocation list, DID-method-native revocation).
4. **Issue** a challenge: send a random nonce (minimum 128 bits of

entropy), encoded as a **lowercase hex string** (32 hex characters for 128 bits). The nonce MUST be transmitted and signed as a UTF-8 encoded string of its hex representation.

5. **Verify** the agent returns a valid Ed25519 signature over the UTF-8 bytes of the hex-encoded nonce string using the key referenced in `authentication`
6. **Confirm** the signing key is not marked `"revoked": true` in the DID Document

Steps 1–6 SHOULD complete within 200 milliseconds for cached DID Documents.

3.2 Pre-Transaction Verification Flow

This section defines the complete verification sequence that a counterparty (verifier) executes before authorizing an agent's requested action. All steps are designed to be executable WITHOUT calling the MolTrust API — this is verifier independence in practice. The verifier needs only the presented credential, a DID resolver, and access to the revocation endpoint.

This flow implements Attribute-Based Access Control (ABAC) as described in NIST SP 800-162 and aligns with the authorization detail structures defined in RFC 9396 (Rich Authorization Requests).

Sequence:

```
Agent                  Counterparty          W3C Resolver
Revocation Registry    AAE Engine         Trust Score
  |                       |                       |
|                    |                  |
  |--- present VC -------->|                       |
|                    |                  |
  |    (AAE embedded)      |                       |
|                    |                  |
  |                       |--- resolve DID ----->|
|                    |                  |
  |                       |<-- DID Document -----|
|                    |                  |
  |                       |                       |
|                    |                  |
  |                       |--- check revocation ------------------
--->|                    |                  |
  |                       |<-- {revoked: false} ------------------
---|                    |                  |
  |                       |                       |
|                    |                  |
  |                       |--- check mandate.allowedActions -------
--->|--- vs requested -->|                  |
  |                       |                       |
|    action           |                  |
  |                       |--- check mandate.deniedActions --------
--->|--- deny override? ->|                  |
  |                       |                       |
|                    |                  |
  |                       |--- check constraints.limits -----------
--->|--- vs tx amount -->|                  |
  |                       |                       |
|                    |                  |
  |                       |--- check
constraints.scope.jurisdictions ->|--- vs context ---->|
|
  |                       |                       |
|                    |                  |
  |                       |--- check constraints.scope -----------
------------------------------------->|
  |                        |    .counterpartyMinScore
|                    |     vs own score |
  |                        |<---------------------------------------
---|--------------------|----- result ----|
  |                        |                      |
|                    |                  |
  |<-- ALLOW or DENY ------|                      |
|                    |                  |
  |    (with reason code)  |                      |
|                    |                  |
```

Verification steps in detail:

1. **Credential presentation:** The agent presents a Verifiable Credential with an embedded AAE (or a standalone AAE credential) to the counterparty.

2. **DID resolution:** The counterparty resolves the agent's DID to its DID Document using a W3C-conformant DID resolver. The counterparty verifies the credential signature against the public key in the DID Document.

3. **Revocation check:** The counterparty queries the `validity.revocationEndpoint` specified in the AAE. If the credential is revoked or the endpoint is unreachable, the request is denied (fail-closed per Section 2.11.4).

4. **Action authorization:** The AAE Engine evaluates `mandate.allowedActions` against the requested action URI. If no pattern matches, the action is denied (default-deny). If a matching `deniedActions` pattern exists, the action is denied regardless of `allowedActions`.

5. **Transaction limits:** The AAE Engine evaluates `constraints.limits` against the transaction parameters. If the amount exceeds `autonomousThreshold`, step-up verification is triggered. If it exceeds `approvalThreshold`, human approval is required.

6. **Jurisdiction check:** The AAE Engine verifies that the transaction context matches `constraints.scope.jurisdictions`. If the counterparty's jurisdiction is not in the allowed list, the action is denied.

7. **Counterparty trust check:** The counterparty evaluates whether the agent meets its own minimum score requirements, AND the agent's AAE evaluates whether the counterparty meets `constraints.scope.counterpartyMinScore`. This is a mutual check.

8. **Decision:** The counterparty returns ALLOW or DENY with a machine-readable reason code.

Reason codes:

Code	Meaning
`allowed`	All checks passed; action is authorized
`denied:credential_expired`	The credential or AAE has expired
`denied:credential_revoked`	The credential or AAE has been revoked
`denied:revocation_unreachable`	The revocation endpoint could not be reached
`denied:action_not_permitted`	The requested action is not in `allowedActions`
	The requested action

`denied:action_explicitly_denied`	matches a `deniedActions` pattern
`denied:limit_exceeded`	The transaction amount exceeds the applicable threshold
`denied:jurisdiction_mismatch`	The transaction context does not match allowed jurisdictions
`denied:counterparty_score_insufficient`	The counterparty's trust score is below the required minimum
`denied:holder_binding_mismatch`	The presenting agent's DID does not match `holderBinding`
`denied:signature_invalid`	The credential signature could not be verified

3.3 Authorization Verification

1. **Request** the agent's AuthorizationCredential for the relevant vertical and action
2. **Verify** the credential signature against the issuer's DID Document
3. **Check** expiry: `expirationDate` MUST be in the future at verification time (UTC)
4. **If delegation chain:** recursively verify each credential from subject to root principal; reject if any link is invalid, expired, or if the chain exceeds 8 hops
5. **Verify** that `permittedActions` includes the claimed action
6. **Verify** that `vertical` matches the context of the interaction
7. **Check** that the issuer DID is not revoked (using a valid revocation source per Section 3.1, step 3)

Verifiers SHOULD cache verified credential chains with a TTL not exceeding 300 seconds.

3.4 Behavioral History Verification

1. **Query** the agent's trust score from a registry endpoint
2. **Verify** the response signature against the registry's published DID
3. **Apply** the score as an advisory input per local policy
4. **Optionally** request and verify individual interaction proofs for spot-checking

Verifiers MUST NOT treat trust scores as authoritative verdicts. A trust score is one input into a local trust decision. The weight given to the score is application-defined.

3.5 Interaction Proof Verification

To verify a submitted interaction proof:

1. **Check** that `id` is not already in the registry (duplicate rejection)
2. **Verify** `proofInitiator` signature against initiator's DID Document
3. **If bilateral:** verify `proofResponder` signature against responder's DID Document
4. **If one-sided:** confirm `"singleSig": true` is present
5. **Check** that `outcome` is a valid value
6. **Confirm** `outcomeHash` is a valid SHA-256 hex string
7. **Check** neither party's DID is revoked

4. Reference Reputation Model (Layer C — Informative)

This section is informative. Implementations MAY use a different scoring model provided they accept Layer A evidence formats. The following model is intentionally simple and illustrative. It is not a mathematically rigorous anti-manipulation guarantee. It is a reasonable heuristic that balances signal richness against complexity.

4.1 Score Range and Grades

Range	Grade	Interpretation
80–100	A	Strong behavioral record, diverse endorsements
60–79	B	Good record, limited cross-vertical coverage
40–59	C	Emerging record, limited history
20–39	D	Thin or inconsistent record
0–19	F	Insufficient data or flags present

Scores are withheld (`null`) until the minimum endorser threshold is reached (Section 4.3).

4.2 Score Formula

```
score = clamp(
  0.6 * direct_score
  + 0.3 * propagated_score
  + 0.1 * cross_vertical_bonus
  + interaction_bonus
  - sybil_penalty,
  0, 100
)
```

`direct_score`

```
direct_score = (sum(w_i * e_i * d_i) / sum(w_i)) * 100
```

Note: weighted mean, not simple mean over n endorsements. This gives higher-trust endorsers proportionally more influence.

- `w_i` = endorser trust score / 100
- `e_i` = endorsement weight declared in credential (0.0–1.0)
- `d_i` = time decay: `exp(-0.005 * age_in_days)`

`propagated_score`

```
propagated_score = mean(trust_score(endorser_i) for all endorsers)
```

Single-hop only. This rewards being endorsed by high-trust agents.

`cross_vertical_bonus`

```
cross_vertical_bonus = min(n_distinct_verticals * 5, 20)
```

Distinct verticals are counted at the `<namespace>/<identifier>` level.

`interaction_bonus`

```
interaction_bonus = min(
  n_bilateral * 0.5 + n_single_sig * 0.2,
  10
)
```

`sybil_penalty`

```
sybil_penalty = 20 * max(0, jaccard(endorsers_A, endorsers_B) - 0.7)
```

Where `endorsers_A` and `endorsers_B` are the endorser DID sets of the scored agent and its most similar peer. Jaccard threshold 0.7 is a heuristic; implementations SHOULD tune this value based on observed network topology. This is not a robust sybil detection system; it is a lightweight signal.

4.3 Minimum Endorser Threshold

Scores are withheld until an agent has endorsements from at least 3 distinct endorser DIDs. Seed agent bootstrap weights count toward this threshold during the bootstrap period only.

4.4 Bootstrap Weight (replaces Seed Agent Floor)

Registry operators MAY register agents with a `bootstrap_weight` — an initial score contribution that decays over time. Bootstrap weight is not a hard floor; it is an additive contribution that diminishes as organic endorsements accumulate.

```
effective_score = computed_score + bootstrap_contribution

bootstrap_contribution = bootstrap_weight * decay_factor

decay_factor = max(0,
  1 - (days_since_registration / bootstrap_period_days)
  - (organic_endorsement_count / bootstrap_endorsement_target)
)
```

Reference values: `bootstrap_period_days = 90`, `bootstrap_endorsement_target = 10`.

This means a seed agent's bootstrap contribution reaches zero after 90 days OR after receiving 10 organic endorsements from non-operator endorsers, whichever comes first. After that point, the agent's score is entirely determined by organic evidence.

Who may be a bootstrap agent: Any agent registered by the registry operator. Operator SHOULD publish the list of bootstrap agents and their bootstrap parameters. Bootstrap endorsements MUST use `"basis": "operator"` and are excluded from diversity calculations that benefit the bootstrap agent itself.

4.5 Behavioral Consistency Signal

Supplementary to the score; SHOULD be exposed alongside it in API responses.

```
consistency = 1 - (std_deviation(outcome_values) / 100)
```

Where `outcome_values`: `completed -> 100`, `partial -> 50`, `disputed -> 10`, `failed -> 0`.

An anomaly is flagged when consistency drops more than 0.3 within any rolling 30-day window relative to the prior 90-day baseline. This is a heuristic signal, not a definitive fraud indicator.

4.6 Score Caching

Cache TTL: 3600 seconds. Cache MUST be invalidated on new endorsement, endorsement expiry, revocation event, or violation recording.

---

5. Reference Registry (Layer B)

5.1 Registry API Endpoints

The reference registry MUST expose the following endpoints:

Endpoint	Method	Description
`/identity/did/{did}`	GET	Resolve DID Document
`/identity/register`	POST	Register new agent DID
`/identity/revoke`	POST	Revoke DID or credential
`/skill/trust-score/{did}`	GET	Query trust score
`/skill/endorse`	POST	Submit endorsement
`/skill/endorsements/{did}`	GET	List endorsements received
`/skill/endorsements/given/{did}`	GET	List endorsements issued
`/interaction/proof`	POST	Submit interaction proof
`/interaction/proofs/{did}`	GET	List proofs for agent
`/violation/record`	POST	Submit violation record (operator only)
`/violation/{id}`	GET	Retrieve violation record
`/revocation/{credential-id}`	GET	Check credential revocation status
`/revocation/status-list/{credential-type}`	GET	Bitstring status list for credential type
`/swarm/stats`	GET	Network-level statistics
`/health`	GET	Registry health status

5.2 Trust Score Response Format

```json
{
  "did": "did:moltrust:<agent-id>",
  "trust_score": 72.4,
  "grade": "B",
  "withheld": false,
  "endorser_count": 5,
  "breakdown": {
    "direct_score": 68.2,
    "propagated_score": 74.1,
    "cross_vertical_bonus": 10.0,
    "interaction_bonus": 3.5,
    "sybil_penalty": 0.0,
    "bootstrap_contribution": 0.0,
    "computation_method": "moltrust-v0.3"
  },
  "consistency": 0.91,
  "anomaly_flag": false,
  "computed_at": "2026-03-22T10:00:00Z",
  "cache_valid_until": "2026-03-22T11:00:00Z",
  "registry_signature": {
    "type": "Ed25519Signature2020",
    "verificationMethod": "did:moltrust:registry#keys-1",
    "proofValue": "<base58btc-signature>"
  }
}
```

All trust score responses MUST be signed by the registry operator key to allow verifiers to confirm the response has not been tampered with in transit. The `computation_method` field identifies the scoring model version used; `moltrust-v0.3` refers to the reference model defined in Section 4 of this specification.

5.3 Revocation

The registry maintains a revocation list as a signed JSON document, updated on every revocation event.

Revocation MUST propagate to verifiers within 60 seconds in the reference implementation via cache invalidation. Verifiers that cache responses MUST honor the `cache_valid_until` field and revalidate on expiry.

The reference implementation does not guarantee instantaneous propagation to independent verifiers. Verifiers with strict security requirements SHOULD perform online verification rather than relying on cached responses.

5.4 Operator Identity

The registry operator MUST publish its own DID Document at a well-known URL:

```
https://<registry-domain>/.well-known/did.json
```

The operator DID is used to sign trust score responses and violation records. Verifiers MUST resolve and cache the operator DID Document before verifying registry-signed artifacts.

---

6. On-Chain Anchoring (Layer B)

6.1 What Is Anchored

Event	Requirement	Data anchored
Agent registration	SHOULD	SHA-256 of DID Document
High-value credential issuance	MAY	SHA-256 of VC
Trust score snapshot	MAY	Score + timestamp hash
Confirmed violation	MUST	SHA-256 of ViolationRecord
Document integrity	SHOULD	SHA-256 of spec or whitepaper

6.2 Anchor Format

```
MolTrust/<event-type>/<version> SHA256:<64-char-hex-hash>
```

Examples:

```
MolTrust/AgentRegistration/1 SHA256:ffbc2b04...
MolTrust/Violation/1 SHA256:3a8f91c2...
MolTrust/DocumentIntegrity/1 SHA256:ffbc2b04...
```

6.3 Verification

Any party MAY verify an anchor by:

1. Recomputing SHA-256 of the relevant artifact
2. Looking up the originating transaction on the chain
3. Decoding the calldata and comparing the hash
4. Verifying the sender address matches the registry's published operator wallet

No proprietary tooling is required beyond a SHA-256 implementation and a public block explorer.

6.4 Chain Requirements

The reference implementation uses Base L2 (mainnet). Implementations MAY use any EVM-compatible L2 with: transaction finality under 10 seconds, permanent data availability, and a public block explorer. The anchor format is chain-agnostic.

---

7. Chain-Agnostic Anchoring (Layer B)

7.1 Overview

As of Protocol v0.7, the MolTrust Protocol supports chain-agnostic anchoring. Proofs MAY be anchored on Base L2 (default), Solana, or SKALE. The chain serves as a timestamp provider, not a trust source. A SHA-256 hash is equivalent on any chain. The protocol does not change — only transport and indexing.

7.2 Anchor Record Schema

All anchor records MUST contain a `chain` field:

```json
{
  "chain": "base",
  "tx": "0xde579d2c...f63d4c",
  "block": 43992036,
  "calldata_prefix": "MolTrust/DID/v1/",
  "anchored_at": "2026-03-29T10:00:00Z"
}
```

Records without a `chain` field: implicitly `chain="base"` (backward compatibility with pre-v0.7 records).

7.3 Universal Calldata Format

UTF-8 string, chain-independent:

```
MolTrust/<type>/<version>/<payload>
```

Format	Example	Bytes
DID Key	`MolTrust/DID/v1/<16hex>/<64hex>`	56

IPR Batch	`MolTrust/IPR/v1/<64hex>`	27
Document	`MolTrust/DocumentIntegrity/1 SHA256:<64hex>`	58

All formats MUST be valid UTF-8 and MUST NOT exceed 566 bytes (Solana Memo limit).

7.4 Chain-Specific Transport

EVM Chains (Base L2, SKALE):

Calldata = UTF-8 string, hex-encoded, submitted as a self-send transaction (from = to = operator wallet). Chain ID is embedded in the EVM transaction signature per EIP-155, preventing cross-chain TX replay.

Solana:

SPL Memo Program. UTF-8 calldata submitted directly. Limit: 566 bytes. All current formats are 27–58 bytes. If a future format exceeds 566 bytes: migration to a Custom Solana Program; Memo remains as fallback for compact formats.

7.5 Supported Chains

Chain	Status	TX Cost	Finality
Base L2	Default	~$0.001	~2s
Solana	Supported (v0.7)	~$0.00025	~400ms
SKALE	Planned	$0.00	~1s
Ethereum L1	Not supported	$5–50	~12s

Chain selection criteria: Sufficient decentralization (no single operator), stable finality (no frequent reorgs), ecosystem relevance for AI agents, technical fit for the calldata format, cost < $0.01 per TX. New chains are selected by MolTrust / CryptoKRI GmbH. Community proposals via GitHub Issues are accepted.

7.6 DID Document — chainAnchor Extension

DID Documents MAY include a `chainAnchor` property (W3C DID Core extension property):

```json
{
  "@context": [
    "https://www.w3.org/ns/did/v1",
    "https://w3id.org/security/suites/ed25519-2020/v1",
    "https://moltrust.ch/ns/chain-anchor/v1"
  ],
  "id": "did:moltrust:abc123",
  "verificationMethod": [{
    "id": "did:moltrust:abc123#key-1",
    "type": "Ed25519VerificationKey2020",
    "controller": "did:moltrust:abc123",
    "publicKeyHex": "<64 hex chars>"
  }],
  "chainAnchor": {
    "chain": "base",
    "tx": "0x...",
    "block": 43992036,
    "calldataFormat": "MolTrust/DID/v1/",
    "rfc8785Canonical": true
  }
}
```

The DID identifier MUST remain chain-agnostic: `did:moltrust:<identifier>`. The chain is recorded in the anchor record, not in the identifier. An agent whose anchor migrates from Base to Solana retains its DID.

7.7 Multi-Chain Verifier

Verifiers MUST support lazy RPC initialization — connections to chain RPCs are established only when a verification request requires them.

```
const verifier = new MolTrustVerifier({
  chains: {
    base:   { rpcUrl: 'https://mainnet.base.org' },
    solana: { rpcUrl: 'https://api.mainnet-beta.solana.com' },
  },
  allowApiFailback: false,  // Default: decentralized, no SPOF
});
```

`allowApiFailback` is opt-in and disabled by default. When enabled, the verifier falls back to the MolTrust API if the chain RPC is unavailable. Results obtained via API fallback MUST be marked as `method: "api-fallback"` with a warning. Without fallback, the verifier returns an honest error (`chain unavailable`) rather than silent degradation.

7.8 Replay Attack Security

- **EVM→EVM replay:** Prevented by EIP-155 chain ID in transaction signature.
- **EVM→Solana replay:** Impossible — incompatible transaction formats.
- **Same calldata, new TX on different chain:** The anchor record contains both `chain` and `tx` — the verifier checks both. A record claiming `chain: "solana"` with a Base TX hash will fail verification (TX not found on Solana).
- **Falsified chain field:** The verifier verifies the TX on the declared chain — a false chain field results in “TX not found.”

No additional replay protection mechanism is required.

7.9 API Endpoints

POST /identity/register — Updated: optional `chain` parameter (default: `"base"`).

```
{
  "display_name": "My Agent",
  "public_key": "<hex>",
  "chain": "solana"
}
```

GET /identity/anchor/:did — Returns all anchor records for a DID.

```
{
  "did": "did:moltrust:abc123",
  "anchors": [
    { "type": "public_key", "chain": "base",   "tx": "0x...",
"block": 43992036 },
    { "type": "public_key", "chain": "solana", "tx": "5Uvw...",
"block": 280000000 }
  ]
}
```

7.10 Database Migration

```
        ALTER TABLE agents
          ADD COLUMN IF NOT EXISTS anchor_chain VARCHAR(20) DEFAULT 'base';

        ALTER TABLE interaction_proof_records
          ADD COLUMN IF NOT EXISTS anchor_chain VARCHAR(20) DEFAULT 'base';

        -- Backfill: all existing records → base
        UPDATE agents SET anchor_chain = 'base' WHERE anchor_chain IS NULL;
        UPDATE interaction_proof_records SET anchor_chain = 'base' WHERE
anchor_chain IS NULL;
```

7.11 Backward Compatibility

All existing anchors on Base L2 remain valid (`anchor_chain='base'` via backfill). All existing DID Documents remain valid (`chainAnchor` is optional). `@moltrust/verify` v1.x continues to function unchanged (Base is the implicit default). New agents MAY choose their chain at registration.

---

8. Credential Lifecycle (Layer A)

7.1 Issuance

Credentials MUST carry: `issuanceDate`, `expirationDate`, valid issuer signature. Maximum validity: 365 days for endorsements, 730 days for authorization credentials. Issuers MUST verify the subject DID before issuance.

7.2 Renewal

Renewal MUST generate a new `id`, new `issuanceDate`, and new `expirationDate`. Renewal does not reset behavioral history.

7.3 Revocation

Any issuer MAY revoke credentials they issued. Revocation is submitted to the registry revocation endpoint with the credential `id` and a revocation signature from the issuer key.

7.4 Expiry

Verifiers MUST reject credentials where `expirationDate` is in the past at verification time (UTC). There is no grace period.

---

9. Agent Lifecycle (Layer A)

8.1 Registration

Registration requires: a conformant DID Document, a principal DID (for non-root agents), an initial AuthorizationCredential. Optional: stake deposit, bootstrap weight (operator-assigned only).

8.2 Bootstrap Period

If a bootstrap weight is assigned at registration, it decays as defined in Section 4.4. After the bootstrap period ends, the agent’s score is entirely organic. The registry MUST expose `bootstrap_contribution` in the trust score breakdown so verifiers can distinguish organic from bootstrapped scores.

8.3 Sub-Agents

Sub-agents MUST have their own distinct DID. They MUST hold an AuthorizationCredential from the parent agent. They do NOT inherit the parent's behavioral history. Maximum delegation depth: 8 hops from root principal.

8.4 Cloning and Redeployment

Clone (same code, new deployment, new operator context): MUST register a new DID. Does NOT inherit history. Representing a clone as the original is a `clone-impersonation` violation.

Redeployment (same logical identity, same principal, new infrastructure): SHOULD use the same DID with key rotation rather than re-registration. Key rotation preserves identity continuity.

8.5 Principal Continuity

Violation Records are associated with both the agent DID and the principal DID. Re-registration of a new agent DID for a principal with confirmed, unresolved violations MUST be flagged by the registry. This association depends on the principal DID being stable; principals SHOULD NOT rotate their DID.

8.6 Deregistration

On deregistration: DID marked inactive, credentials remain valid until their own expiry, behavioral record retained per Section 10.4, stake returned if no unresolved violations.

8.7 Optional Stake

Agents MAY deposit USDC stake in a registry smart contract at registration. Stake is returned on clean deregistration and forfeited on confirmed violation. Minimum stake for the signal to be meaningful: 10 USDC (reference value; operator-defined). Stake forfeiture requires a ViolationRecord (Section 2.7); unilateral accusation does not trigger forfeiture.

---

10. Threat Model

9.1 Scope

This section covers known attack vectors. It is not exhaustive. New attack classes will emerge as the agent economy develops.

9.2 Attack Summary

Attack	Mitigations	Residual Risk
Sybil clusters	Cross-vertical requirement, Jaccard detection, stake cost, x402 economic barriers (Section 2.10)	Well-funded patient attacker can still construct convincing clusters

Slow-burn trust accumulation	Consistency signal, stake forfeiture, endorsement decay (Section 2.10.4)	Patient attacker with low-detectable violation may not trigger signal
Key theft / impersonation	Key rotation, revocation propagation, consistency discontinuity	Stale-cache verifiers may not detect revocation in time
Collusion / bribed endorsements	Endorser weight propagation, Jaccard detection	High-trust collusion is harder to detect
Reputation laundering	Principal DID continuity, on-chain violation permanence	Principal with new identity cannot be auto-linked without external evidence
Replay attacks	UUID deduplication, expiry dates, challenge nonces	None for conformant implementations
Data withholding	One-sided proofs, one-sided proof patterns	Bilateral collusion to suppress proofs is undetectable
Hash preimage inference	Outcome hash includes low-entropy fields only	For predictable outcomes, hash may be correlatable
Endorsement farming	Basis field, evidence hash requirement, one-per-vertical rule	Subjective endorsements without interaction proof basis are harder to detect
Adjudicator compromise	ViolationReversal mechanism, operator-signed records	Corrupted external adjudicator could produce false violations
AAE bypass / privilege escalation	Default-deny, deny precedence, holder binding, attenuation-only delegation (Section 2.8)	Implementation bugs in AAE evaluation logic

9.3 Notes on Jaccard Sybil Detection

The Jaccard similarity threshold (0.7) is a lightweight heuristic suitable for small-to-medium networks. It does not catch sophisticated sybil clusters that deliberately diversify their endorser sets. Operators running large networks SHOULD supplement Jaccard detection with graph-theoretic clustering algorithms. This is outside the scope of this specification.

---

11. Privacy Model

10.1 Principles

Data minimization: Only hashes and structural metadata are submitted to shared infrastructure. Raw transaction content, counterparty details, and outcome specifics are retained locally by the parties.

Pseudonymity by default: DID Documents contain no personal data. Identity binding between a DID and a natural person is external to this protocol.

Separation of on-chain and off-chain data: On-chain anchors contain only hashes. An observer of on-chain data learns that an event occurred — not what it contained.

10.2 What Is Stored Where

Data element	Storage	Accessible to
DID Document	Registry (off-chain)	Public
Authorization credential	Agent + Registry	Verifiers on request
Interaction proof structure	Registry (off-chain)	Verifiers on request
Outcome hash	Registry (off-chain)	Public
Raw outcome data	Local only	Parties to the interaction
On-chain anchor hash	Blockchain	Public, permanent
Trust score	Registry (off-chain)	Public
Endorsement	Registry (off-chain)	Public
Stake balance	Blockchain	Public
Violation Record	Registry (off-chain) + on-chain hash	Public
Tier 0 credential	Agent + Registry	Verifiers on request
AAE	Agent + Counterparty	Presented per transaction

10.3 Hash Preimage Risks

`outcomeHash` is a SHA-256 hash of a structured payload. For interactions with low-entropy or predictable outcomes (e.g. a binary success/failure), the hash may be correlatable by an attacker who can enumerate possible outcomes. Parties handling sensitive interactions SHOULD include a random salt in the outcome payload before hashing to prevent preimage correlation.

10.4 GDPR and Swiss DSG Considerations

This analysis is informational and does not constitute legal advice.

Personal data: A conformant DID Document contains no personal data. If a DID can be linked to a natural person through external means, interaction records involving that DID may constitute personal data processing under GDPR Article 4 and Swiss DSG.

Right to erasure: On-chain anchors are permanent and cannot be deleted. Implementers MUST NOT anchor personal data or pseudonymous identifiers that can be directly linked to a natural person. Hashes of non-personal protocol artifacts (DID Documents, ViolationRecords) are compatible with GDPR compliance as they do not directly identify individuals.

Data retention: Off-chain behavioral records SHOULD be retained for a minimum of 12 months (dispute resolution) and a maximum of 60 months, after which they SHOULD be deleted unless required by applicable law.

Data Processing Agreements: Organizations using the MolTrust reference API to process data relating to natural persons MUST establish a Data Processing Agreement with MolTrust / CryptoKRI GmbH.

Tier 0 privacy: DeveloperIdentityCredential (Section 2.9) does NOT contain personal data. The credential attests that KYC verification occurred; the personal data used during verification is held exclusively by the KYC provider under a separate data processing relationship.

10.5 Future: Zero-Knowledge Extensions

ZK-proof techniques (e.g. zk-SNARKs) could allow agents to prove behavioral properties without revealing underlying interaction proofs. This is not specified in v0.3 and is deferred to a future extension.

---

12. Worked Example

Scenario: A travel booking agent (Agent A) wants to book a hotel through Agent B. Agent B requires trust score >= 60.

Step 1 — Identity Verification

Agent B sends nonce `"f7a3c2d9"` to Agent A.

Agent A signs the nonce with its Ed25519 key and returns:

```
{
  "did": "did:moltrust:traveler-agent-001",
  "nonce": "f7a3c2d9",
  "signature": "<Ed25519 signature over UTF-8 nonce bytes>"
}
```

Agent B resolves the DID, retrieves the public key, verifies the signature.

Step 2 — Authorization Verification

Agent A presents its AuthorizationCredential:

```
{
  "type": ["VerifiableCredential", "AuthorizationCredential"],
  "issuer": "did:moltrust:human-traveler-principal",
  "credentialSubject": {
    "id": "did:moltrust:traveler-agent-001",
    "permittedActions": ["transact"],
    "vertical": "moltrust/travel",
    "constraints": { "maxTransactionValue": 2000 }
  }
}
```

Agent B verifies: signature valid, not expired, `transact` in `permittedActions`, vertical matches.

Step 3 — Behavioral History

Agent B queries:

```
GET https://api.moltrust.ch/skill/trust-score/did:moltrust:traveler-agent-001
```

Response: `trust_score: 72.4`, `grade: B`, `withheld: false`, signed by registry. Score >= 60.

Step 4 — Interaction

Agent A submits booking request. Agent B confirms reservation.

Step 5 — Interaction Proof

Agent A constructs and signs:

```
        {
          "type": "InteractionProof",
          "id": "550e8400-e29b-41d4-a716-446655440000",
          "session": "booking-2026-0322-001",
          "initiator": { "did": "did:moltrust:traveler-agent-001",
"vertical": "moltrust/travel" },
          "responder": { "did": "did:moltrust:hotel-agent-002", "vertical":
"moltrust/travel" },
          "timestamp": "2026-03-22T14:30:00Z",
          "outcome": "completed",
          "outcomeHash": "sha256:9f86d081...",
          "proofInitiator": { "proofValue": "<Agent A signature>" }
        }
```

Agent B adds `proofResponder` and submits to registry. Both agents' behavioral records are updated at the next scoring cycle.

---

13. Conformance

12.1 Layer A — Protocol Conformance

A Layer A conformant implementation MUST:

1. Issue DIDs conforming to W3C DID Core v1.0
2. Issue and verify credentials conforming to W3C VC Data Model 2.0
3. Produce interaction proofs containing all mandatory fields defined in Section 2.4
4. Sign all artifacts using Ed25519Signature2020 over RFC 8785 canonical JSON (excluding proof fields)
5. Use UUID v4 for all `id` fields
6. Reject credentials with `expirationDate` in the past
7. Reject interaction proofs with duplicate `id` values
8. Enforce delegation chain depth limit of 8 hops
9. Express verticals using the `<namespace>/<identifier>` format defined in Section 2.5
10. Produce endorsements conforming to Section 2.6
11. Produce violation records conforming to Section 2.7 when recording violations
12. Enforce AAE validation rules defined in Section 2.8.4 when processing Agent Authorization Envelopes
13. Implement credential TTL enforcement and revocation checking per Section 2.11

12.2 Layer B — Registry Conformance

A Layer B conformant registry MUST:

1. Implement all endpoints defined in Section 5.1
2. Return trust score responses in the format defined in Section 5.2
3. Sign all trust score responses with the operator registry key
4. Maintain a revocation list and propagate revocations within 60

seconds
5. Publish the operator DID Document at `/.well-known/did.json`
6. Associate ViolationRecords with both agent DID and principal DID
7. Record confirmed violations on-chain per Section 6
8. Expose revocation endpoints per Section 2.11.2 and Bitstring Status Lists per Section 2.11.3

12.3 Layer C — Reputation Model Conformance

There is no mandatory conformance requirement for Layer C. Implementations MAY use any scoring model that consumes Layer A evidence. Implementations that use the reference model SHOULD implement all components defined in Section 4.

12.4 Non-Goals

A conformant implementation is NOT required to:

- Adjudicate disputes
- Evaluate agent output quality
- Enforce legal compliance
- Interoperate with non-conformant implementations

12.5 Version Compatibility

Version 0.5 is the current specification. Breaking changes to Layer A data formats will carry a minimum 12-month deprecation period in future versions. Section 7 (Chain-Agnostic Anchoring) is additive — all pre-v0.5 anchor records remain valid with implicit `chain="base"`. Layer B API changes follow semantic versioning. Layer C changes are non-breaking by definition.

References

- W3C DID Core v1.0: https://www.w3.org/TR/did-core/
- W3C VC Data Model 2.0: https://www.w3.org/TR/vc-data-model-2.0/
- W3C Bitstring Status List v1.0: https://www.w3.org/TR/vc-bitstring-status-list/
- Ed25519Signature2020: https://w3c-ccg.github.io/di-eddsa-2020/
- RFC 8785 (JSON Canonicalization): https://www.rfc-editor.org/rfc/rfc8785
- RFC 2119 (Key Words): https://www.rfc-editor.org/rfc/rfc2119
- RFC 4122 (UUID): https://www.rfc-editor.org/rfc/rfc4122
- RFC 9396 (Rich Authorization Requests): https://www.rfc-editor.org/rfc/rfc9396
- NIST SP 800-162 (Guide to ABAC): https://csrc.nist.gov/pubs/sp/800/162/final
- EIP-155 (Chain ID for TX Replay Protection): https://eips.ethereum.org/EIPS/eip-155
- SPL Memo Program: https://spl.solana.com/memo
- ERC-8004: https://eips.ethereum.org/EIPS/eip-8004
- Trusted Agentic Mesh (TAM): https://www.ijfmr.com/papers/2026/1/66724.pdf
- AgentHub — Agent Registry and Provenance: https://arxiv.org/abs/2510.03495
- W3C AI Agent Protocol Community Group: https://agent-network-protocol.com
- DIF Trusted AI Agents Working Group: https://identity.foundation
- MolTrust Reference Implementation: https://api.moltrust.ch
- MolTrust Protocol Whitepaper: https://moltrust.ch/whitepaper

---

MolTrust / CryptoKRI GmbH, Zurich api.moltrust.ch · moltrust.ch · info@moltrust.ch

---