The MolTrust Protocol: Technical Specification

Version 0.10 — Draft for Review

MolTrust / CryptoKRI GmbH, Zurich

October 2026

Contents

The MolTrust Protocol: Technical Specification 1
Version 0.10 — Draft for Review 1
Architectural Overview 2
Table of Contents 2
1. Scope and Terminology 4
2. Data Model (Layer A) 6
3. Verification Flow (Layer A) 30
4. Reference Reputation Model (Layer C — Informative) 33
5. Reference Registry (Layer B) 36
6. On-Chain Anchoring (Layer B) 39
7. Credential Lifecycle (Layer A) 39
8. Cross-Protocol Interoperability 40
9. Infrastructure-Layer Enforcement 44
10. Governance Layer 48
11. Outcome Verification 49
12. Agent Lifecycle (Layer A) 50
13. Threat Model 51
14. Privacy Model 53
15. Worked Example 54
16. Conformance 55
17. Enforcement Layer & Governance Transition 57
18. Leak Canary 60
References 64

The MolTrust Protocol: Technical Specification

Version 0.10 — Draft for Review

MolTrust / CryptoKRI GmbH, Zurich October 2026 Status: Informational Draft

v0.7 additions: Cross-Protocol Interoperability (qntm/APS), Infrastructure-Layer Enforcement (Falco), Governance Layer, Outcome Verification.

v0.8.1 additions: A2A v0.3 Conformance (Sec. 8.8) v0.9 additions: Enforcement Layer & Governance Transition (Sec. 17) — AAE enforcement layer (advisory) + CEP governance-transition model (de-

signed, not activated) v0.10 additions: Leak Canary (Sec. 18) — per-counterparty canary credentials, modes `bound` and `decoy`, signed leak events with holder-proof attribution (live)

This document is a companion to *The MolTrust Protocol: A Verification Standard for Autonomous Software Agents* (Whitepaper v0.8). 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
   - 2.12 Multi-Chain Wallet Binding

- 2.13 External DID Bridging
- 2.14 Cross-Ecosystem Trust Score Import

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. Credential Lifecycle (Layer A)
8. Cross-Protocol Interoperability
   - 8.1 AAE ↔ qntm Authority Constraints Mapping
   - 8.2 AAE ↔ APS Delegation Mapping
   - 8.3 decision-equivalence Layer
9. Infrastructure-Layer Enforcement
   - 9.1 Architecture
   - 9.2 Webhook Payload Schema
   - 9.3 Falco Rule Design 9.4 Sequential Action Safety (SAS)
10. Governance Layer

- 10.1 Layer Classification
- 10.2 aps.txt Security Analysis

11. Outcome Verification

- 11.1 FlagRecord Schema
- 11.2 OutcomeRecord Schema
- 11.3 FlagScore Formula
- 11.4 Settlement Watcher

12. Agent Lifecycle (Layer A)
13. Threat Model
14. Privacy Model
15. Worked Example
16. Conformance
17. Enforcement Layer & Governance Transition

- 17.1 Enforcement Layer (implemented, advisory)
- 17.2 Governance Transition (CEP, designed)
- 17.3 Summary of status

18. Leak Canary

- 18.1 A canary is an ordinary credential
- 18.2 Modes

- 18.3 Flow
- 18.4 Attribution
- 18.5 Leak event
- 18.6 Privacy
- 18.7 Limits
- 18.8 Summary of status

---

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 12.2).

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.

Wallet Binding — a cryptographic proof that an agent controls a specific blockchain wallet, linking the wallet address to the agent's DID.

DID Bridge — a verified link between an external DID (from another ecosystem) and a `did:moltrust` identity, enabling cross-ecosystem trust portability.

External Score Import — a mechanism for importing reputation scores from external systems into the MolTrust trust score, with reduced weight and accelerated decay.

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:

```
{
  "@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

```
{
  "@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.
`attenuationOnly`	boolean	`true`	If `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.

```
{
  "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/tier
0"
  },
  "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 12.5) 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).

2.12 Multi-Chain Wallet Binding

MolTrust supports wallet binding across multiple blockchain ecosystems. The `chain` parameter in `/identity/nonce` and `/identity/bind` determines the signature scheme used for verification.

Supported chains:

Chain	Signature Scheme	Address Format
`ethereum` (default)	EIP-191 secp256k1	`0x` hex (20 bytes)
`base`	EIP-191 secp256k1	`0x` hex (20 bytes)
`solana`	Ed25519 via PyNaCl	Base58 public key

Nonce request (extended):

```
GET /identity/nonce?did={did}&chain=solana

{
  "nonce": "<random-hex>",
  "chain": "solana",
  "message": "MolTrust wallet binding\nDID: {did}\nNonce: <nonce>",
  "instructions": "Sign this message with your Solana wallet (ed25519)"
}
```

Bind request (Solana):

```
POST /identity/bind
{
  "did": "did:moltrust:<id>",
  "wallet_address": "<base58-pubkey>",
  "signature": "<base58-ed25519-signature>",
  "chain": "solana"
}

Response:
{
  "success": true,
  "did": "did:moltrust:<id>",
  "wallet": "<base58-pubkey>",
  "chain": "solana",
  "payment_ready": true
}
```

DID Document update: Upon successful binding, the agent's DID Document is extended with a chain-specific payment service:

```
{
  "service": [{
    "id": "did:moltrust:<id>#payment",
    "type": "SolanaPaymentService",
    "serviceEndpoint": "<base58-pubkey>"
  }]
}
```

For Ethereum/Base bindings, the service type is `EthereumPaymentService` with a `0x`-prefixed address.

Verification rules: - Each DID MAY have at most one wallet binding per chain. - Binding a new wallet on the same chain replaces the previous binding. - The nonce MUST be used within 300 seconds. - Signature verification uses the chain-native algorithm — there is no cross-chain signature compatibility.

2.13 External DID Bridging

MolTrust supports linking external DID methods to `did:moltrust` identities. This allows agents from other ecosystems to use MolTrust trust infrastructure without abandoning their existing identity.

Supported external DID methods: Any DID method that supports Ed25519 or secp256k1 signing.

Bridge request:

```
POST /identity/bridge
{
  "external_did": "did:<method>:<id>",
  "moltrust_did": "did:moltrust:<id>",
  "proof": "<signature-base58>",
  "chain": "solana" | "ethereum",
  "wallet_address": "<pubkey>"
}

Response:
{
  "success": true,
  "external_did": "did:<method>:<id>",
  "moltrust_did": "did:moltrust:<id>",
  "chain": "solana"
}
```

Prerequisite: The `wallet_address` MUST already be bound to `moltrust_did` via `/identity/bind`. The proof is a signature over the string `"DID Bridge: {external_did} -> {moltrust_did}"` using the bound wallet's private key.

Resolution:

```
GET /identity/resolve-external/{external_did}

{
  "external_did": "did:<method>:<id>",
  "moltrust_did": "did:moltrust:<id>",
  "document": { ... MolTrust DID Document ... }
}
```

Constraints: - Each external DID MAY be bridged to at most one `did:moltrust` identity. - Bridging is not transitive: if A bridges to B and B bridges to C, resolving A does not return C. - Bridge records are stored in the registry and are publicly queryable. - Revoking a wallet binding implicitly invalidates all bridges that depend on that wallet.

2.14 Cross-Ecosystem Trust Score Import

Agents with verified reputation scores in external systems may import those scores as a basis for their MolTrust trust score.

Import request:

```
POST /identity/import-score
{
  "moltrust_did": "did:moltrust:<id>",
  "external_did": "did:<method>:<id>",
  "external_score": 0.83,
  "external_system": "peer_attestation",
  "proof": "<signature>"
}

Response:
{
  "moltrust_did": "did:moltrust:<id>",
  "external_score": 0.83,
  "mapped_score": 91.5,
  "source": "peer_attestation"
}
```

Score mapping: External scores (normalized to 0.0–1.0) are mapped to the MolTrust trust score range (0–100) using logarithmic scaling. External endorsements carry a reduced weight of 0.3 relative to native MolTrust endorsements (weight 1.0).

Mapping formula:

```
mapped_score = min(100, external_score * 100 * external_weight)
external_weight = 0.3  (default for imported scores)
```

Authentication: The `proof` field MUST be a signature over `"Score Import: {external_did} -> {moltrust_did} score={external_score}"` from the wallet bound to the external DID via `/identity/bridge`.

Constraints: - Score imports require an active DID bridge (Section 2.13). - Each external system may contribute at most one score per `moltrust_did`. - Imported scores decay at 2x the normal rate (half-life of 45 days vs. 90 days for native scores). - The `external_system` field is a free-form string for the importing party to self-declare; the registry does not validate external system claims.

---

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

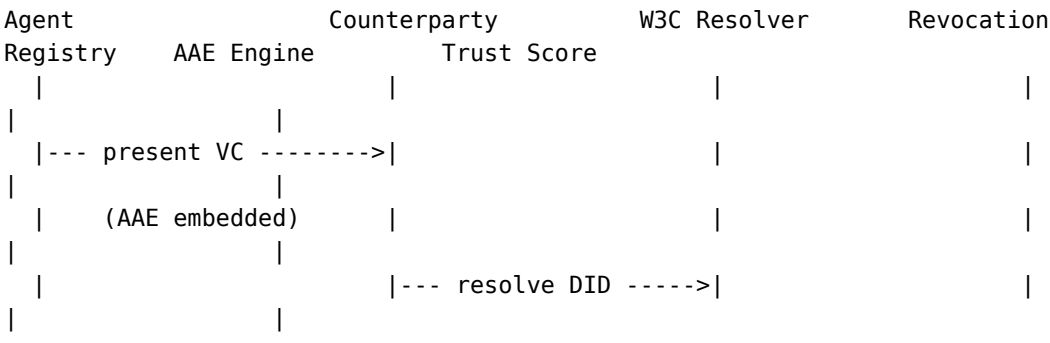
Steps 1–6 SHOULD complete within 200 milliseconds for cached DID Documents.

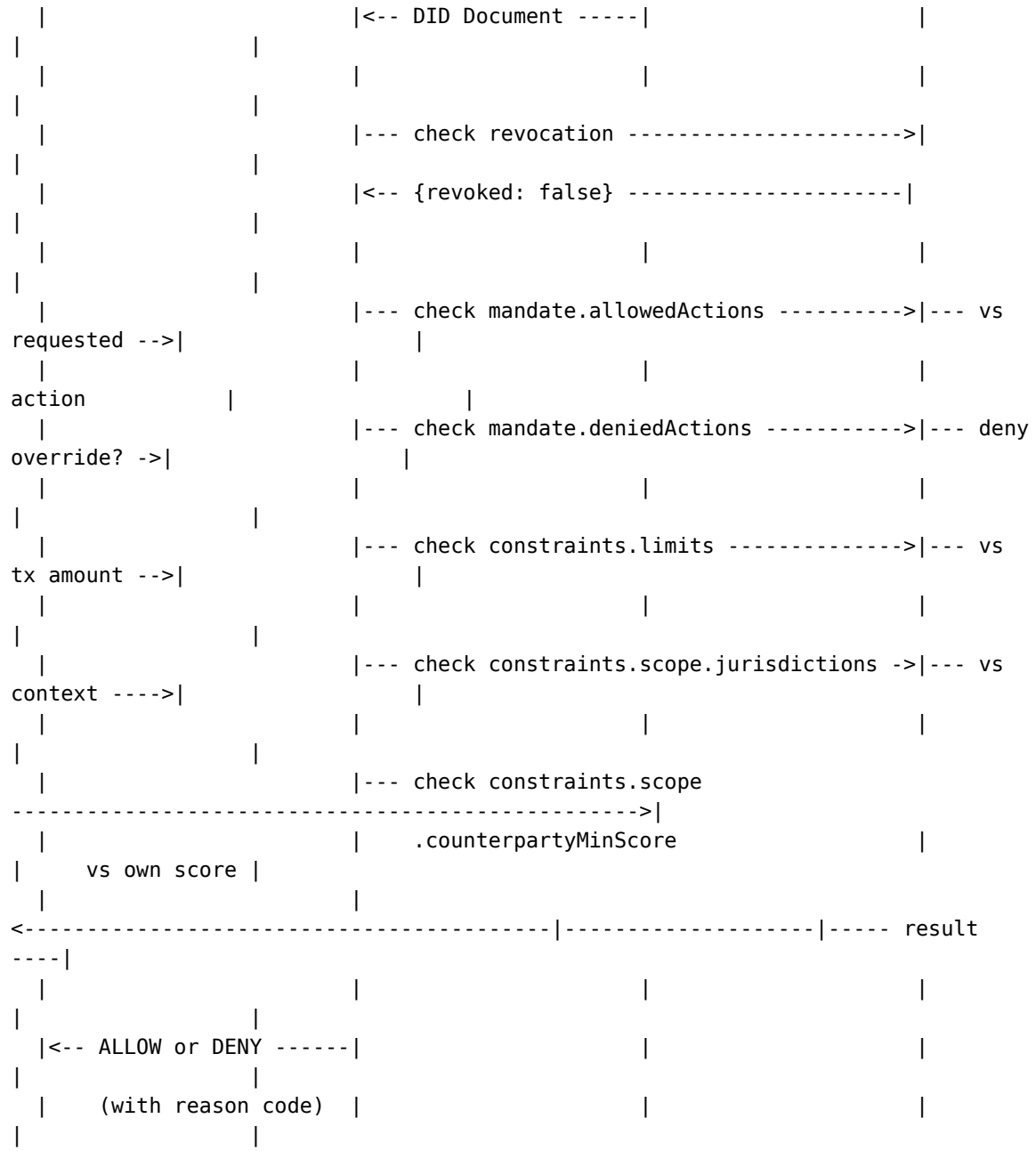
3.2 Pre-Transaction Verification Flow

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:





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`
`denied:action_explicitly_denied`	The requested action 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
`/identity/nonce`	GET	Request nonce for wallet binding (supports `chain` param)
`/identity/bind`	POST	Bind wallet to DID (Ethereum, Base, Solana)
`/identity/bridge`	POST	Bridge external DID to MolTrust DID
`/identity/resolve-external/{did}`	GET	Resolve external DID to MolTrust identity
`/identity/import-score`	POST	Import external trust score (requires bridge)

5.2 Trust Score Response Format

```
{
  "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-v0.7"
  },
  "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-v0.7` 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

6.2 Anchor Format

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

Examples:

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

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. 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.

---

8. Cross-Protocol Interoperability

8.1 AAE ↔ qntm Authority Constraints Mapping

MolTrust AAE fields map directly to the qntm ConstraintEvaluation facets. This mapping was verified through 5 test vectors (TV-001 through TV-005, corpollc/qntm PR #10), independently confirmed by two external implementations.

AAE Field	qntm Facet	Mapping
mandate.purpose + allowedActions	scope	Direct: action URI patterns map to qntm scope constraints
constraints.limits.autonomousThreshold	spend	Direct: USDC threshold maps to qntm spend limit
constraints.duration.ttl	time	Direct: seconds TTL maps to qntm time constraint
constraints.scope.counterpartyMinScore	reputation	Direct: 0-100 score maps to qntm reputation threshold
mandate.delegation.allowed	reversibility	Inverse: delegation=false implies non-reversible commitment

Shared canonicalization: JCS RFC 8785. Shared signing: Ed25519. This means a single signing operation can produce artifacts verifiable by both MolTrust and qntm conformant implementations.

Test vectors TV-001 through TV-005 cover AAE delegation narrowing — ensuring sub-agent AAEs are strict subsets of parent AAEs. All 5 vectors verified by external conformance runners.

8.2 AAE ↔ APS Delegation Mapping

The Agent Provider Service (APS) protocol uses a provider attestation model. MolTrust AAE maps to APS as follows:

- `importProviderAttestation()` accepts MolTrust VCs as attestation input
- AAE trust score (0-100) maps to APS Grade (0-3): Grade 0 = score < 25, Grade 1 = 25-50, Grade 2 = 50-75, Grade 3 = 75-100
- Provider Registry integration is in progress

8.3 decision-equivalence Layer

Cross-system interoperability requires a shared model for comparing agent decisions across protocol boundaries. The decision-equivalence model defines three levels:

1. **Correlation** — `action_ref`: A temporal binding that links an action in one system to an action in another. Implemented as a shared UUID or hash reference.
2. **Equivalence** — Canonical outcome hash via JCS: A semantic binding that proves two systems reached the same conclusion about an interaction outcome. *Planned for TechSpec v0.8.*
3. **Explanation** — System-specific evidence: Each system provides its own proof format (MolTrust: IPR, qntm: ConstraintEvaluation, APS: ProviderAttestation).

MolTrust currently implements Level 1 (via IPR `session` field) and Level 3 (via IPR evidence). Level 2 (canonical outcome hash) is identified as a gap and planned for TechSpec v0.8.

8.4 MPP Interoperability

Status: Live – April 2026 **npm:** `@moltrust/mpp` v1.0.3

The Machine Payments Protocol (MPP), co-authored by Stripe and Tempo, uses the same HTTP 402 challenge-response pattern as x402 but extends it with session-based payments, card support via Stripe SPTs, and Bitcoin Lightning support.

MPP endpoints identify paying agents via an `Authorization: Payment <credential>` header where the credential is a base64-encoded JSON object. The paying agent's wallet address is extracted from `credential.source` (format: `chainId:0xAddress`).

Middleware Integration:

```
const { requireScore } = require('@moltrust/mpp');
app.use(requireScore({ minScore: 60 }));
app.use(mppx.charge({ amount: '0.01' }));
```

Both `@moltrust/x402` and `@moltrust/mpp` expose an identical API: `requireScore({ minScore, failBehavior })`. Endpoint operators can use both simultaneously for multi-protocol support.

8.5 Trust Score Specification

8.5.1 Shadow Score (Unregistered Agents)

Agents without a MolTrust DID receive a shadow score derived from on-chain transaction history:

```
shadow_score = 25 (base)
             + min(10, tx_count * 0.5)
             + min(5,  total_usdc * 0.1)
```

`Range: 25-40. Capped at 40 without DID registration.`

A shadow score of 25 means zero on-chain activity. A score of 40 means 20+ transactions with meaningful USDC volume. Endpoint operators using `requireScore({ minScore: 60 })` will block all unregistered agents, incentivizing DID registration.

8.5.2 Full Trust Score (Registered Agents)

See TechSpec Section 4 (Reference Reputation Model) for the full Phase 2 formula. Cache: PostgreSQL `trust_score_cache` with 1-hour TTL, invalidated on endorsement, revocation, or violation events.

8.6 Security Model

8.6.1 API Authentication

Trust score queries to `/wallet/{address}` are public (no API key required) by design – trust scores are intended to be publicly verifiable, consistent with the W3C VC principle of open verifiability.

Rate limit: 30 requests/minute per IP on `/wallet/{address}`.

8.6.2 Failure Behavior

Both `@moltrust/x402` and `@moltrust/mpp` implement configurable failure behavior:

```
// Default: fail-open (API downtime does not block payments)
app.use(requireScore({ minScore: 60, failBehavior: 'open' }));

// Opt-in: fail-closed (API downtime blocks all requests)
app.use(requireScore({ minScore: 60, failBehavior: 'closed' }));
```

Scenario	fail-open (default)	fail-closed
API reachable, score OK	Pass	Pass
API reachable, score low	403	403
API unreachable	Pass + failOpen flag	403

8.6.3 did:moltrust: Method

`did:moltrust:` is a registry-based DID method. DIDs follow `did:moltrust:<16-hex-char>`. Resolution: `GET https://api.moltrust.ch/identity/did/{did}`. Ed25519 public keys, optional service endpoints. W3C DID Method Registry submission planned for v0.9.

8.7 Performance

Metric	Value
Trust score cache	PostgreSQL, 1h TTL
Rate limit (/wallet/)	30 req/min per IP
Middleware latency (cache hit)	<10ms
Middleware latency (cache miss)	~100-200ms
Horizontal scaling	Planned (Redis cache layer)

8.8 A2A v0.3 Conformance

Status: Live — April 2026

MolTrust exposes a fully A2A v0.3 conformant Agent Card at:

- `https://api.moltrust.ch/.well-known/agent-card.json`
- `https://api.moltrust.ch/.well-known/agent.json` (alias)
- `https://moltrust.ch/.well-known/agent-card.json`

The Agent Card declares five skills (trust-score, did-resolution, credential-verification, wallet-binding, sybil-detection), a MolTrust-specific trust-score extension (`https://moltrust.ch/extensions/trust-score/v1`), and two security schemes (apiKey + X-MolTrust-DID header).

A2A v0.3 Conformance Checklist: - [x] name, description, url, version fields present - [x] provider.organization (CryptoKRI GmbH) - [x] capabilities.extensions (trust-score/v1) - [x] securitySchemes (apiKey + moltrust DID header) - [x] skills array with id, name, description, tags per skill - [x] defaultInputModes / defaultOutputModes

The MolTrust trust-score extension enables any A2A-compatible agent or orchestrator to query trust scores and verify agent credentials directly from the Agent Card metadata, without prior knowledge of MolTrust's API structure.

Trust Score Integration for A2A Clients:

```
GET https://api.moltrust.ch/skill/trust-score/{did}
Header: X-MolTrust-DID: did:moltrust:<requester-id>
```

A2A clients MAY gate task acceptance on counterparty trust score by including a minimum score requirement in task metadata:

```
{
  "metadata": {
    "moltrust_min_score": 60,
    "moltrust_requester_did": "did:moltrust:<id>"
  }
}
```

The MolTrust registry will evaluate the requesting agent's trust score and return payment_ready status alongside the score, enabling combined trust + payment verification in a single API call.

Relationship to AIP (arXiv:2603.24775): MolTrust implements three of the four criteria identified in Prakash (2026) as missing from existing agent identity protocols: offline attenuable delegation (AAE), expressive chained policy (delegation chain traversal), and provenance-aware completion records (IPR + on-chain anchoring). MCP and A2A wire format bindings for AAE tokens are deferred pending stabilization of the A2A authorization scheme roadmap item.

9. Infrastructure-Layer Enforcement

9.1 Architecture

MolTrust defines three enforcement layers for AAE constraints:

```
Layer 1 — Cryptographic
  Ed25519 signatures, JCS RFC 8785 canonicalization
  AAE boundaries are tamper-proof by construction
  Verifiable by any party without infrastructure dependency

Layer 2 — API
  MolTrust Trust Score degradation on violation
  IPR (Interaction Proof Record) submission
  Verifiable Credential revocation

Layer 3 — Kernel (in progress)
  Falco eBPF/syscall detection
  Linux kernel-level constraint monitoring
  Not bypassable by the agent process itself
```

Layer 3 is a qualitative difference: Layers 1 and 2 rely on the agent's runtime environment to report violations. Layer 3 operates below the agent's process boundary — the agent cannot suppress or modify the detection signal.

Status: Layer 3 is live (April 2026). Reference implementation: https://github.com/HaraldeRoessler/moltrust-falco-bridge

9.4 Sequential Action Safety (SAS)

Status: Live — Phase 1 (WARN-only) **Deployed:** April 2026

Overview

MoltGuard evaluates each agent action individually against the AAE envelope (scope, spend, validity). SAS adds a pre-execution check for **order-sensitive action sequences** — detecting cases where individually permitted actions produce catastrophic outcomes when executed in the wrong order.

Example:

```
Action A: DELETE /data/customers/backup   → AAE permits ✅
Action B: WRITE  /data/customers/export   → AAE permits ✅

A before B = data loss (irreversible) ❌
B before A = safe ✅
```

Both actions pass AAE authorization. SAS detects the order dependency.

Safety Residual Formula

The core metric is the **Safety Residual** — an asymmetric measure of whether a proposed action is more destructive than a prior action on overlapping resources:

```
R = max(0, Φ(proposed) - Φ(past)) × overlap(proposed, past)
```

Where `Φ(action)` is the reversibility score of the action type:

Action	Φ (Reversibility)	Rationale
DELETE, PURGE, REVOKE, TERMINATE	1.0	Irreversible
TRANSFER	0.95	USDC on-chain — practically irreversible
WRITE	0.6	Partially reversible
UPDATE	0.5	Partially reversible
READ, LIST, QUERY	0.0	Fully reversible

The `overlap` function measures path similarity, stopping at the first divergence:

```
/data/customers/backup vs /data/customers/export → 2/3 = 0.667
/api/v1/users vs /api/v1/orders → 0.0 (diverges at segment 3)
```

Special rule: DELETE/PURGE/TERMINATE after READ/WRITE on the same resource produces a residual capped at 0.9 with a 2x multiplier, reflecting the high risk of destroying data that was just accessed.

Thresholds and Verdicts

Residual	Verdict	Phase 1 (current)	Phase 2
0.0 – 0.29	SAFE	Execute	Execute
0.3 – 0.69	WARN	Execute + IPR record	Execute + IPR record
0.7 – 1.0	BLOCK	Execute + IPR record	Block + arbitration

Phase 1 operates in WARN-only mode. All actions execute, but order-sensitive sequences are flagged and recorded as IPR events on Base L2.

API

```
POST /guard/api/action/check
{
  "did": "did:moltrust:<id>",
  "session_id": "sess_abc123",
  "proposed_action": {
    "type": "DELETE",
```

```
    "resource": "/data/customers/backup",
    "scope": "data:write"
  }
}
```

The `session_id` is required for pairwise checks. Without it, each action is evaluated in isolation. Sessions expire after 1 hour.

Position in the Stack

```
L1    Identity Check      (AgentID / did:moltrust:)
L2    AAE Authorization   (MANDATE + CONSTRAINTS + VALIDITY)
L2.5  SAS Pre-Execution   (Sequential Action Safety) ← opt-in
L3    Trust Score Gate     (MolTrust Score ≥ threshold)
L4    Falco Enforcement    (Kernel-level, Section 9.1–9.3)
L5    IPR Anchor           (Base L2 non-repudiation)
```

SAS is opt-in. Agents that do not call the endpoint are not checked. SAS does not replace AAE authorization — it supplements it with sequence-aware safety analysis.

Design Constraints

- Deterministic: no LLM calls, stdlib-only computation
- Pairwise: checks proposed vs. each prior action, not N-action optimization
- Ephemeral: session state is in-memory with 1h TTL
- Non-blocking (Phase 1): WARN events are recorded but do not prevent execution

Deployed on OpenClaw gateway instance (prod). Test DID did:moltrust:662a7181e0154998 demonstrates trust score degradation on Falco-detected violations.

9.2 Webhook Payload Schema

When Falco detects a constraint violation, it sends a webhook to the MolTrust IPR endpoint:

```
{
  "output_hash": "sha256:<hash-of-evidence>",
  "agent_did": "did:moltrust:<id>",
  "output_type": "generic",
  "source_refs": ["falco:<rule_name>"],
  "confidence": 1.0,
  "confidence_basis": "declared",
  "produced_at": "<ISO-8601>",
  "metadata": {
    "source": "falco",
    "action": "<denied_action_from_aae>",
    "result": "VIOLATION",
    "rule": "<falco_rule_name>",
    "syscall": "<syscall>",
    "file_path": "<path>",
    "container_id": "<k8s-pod-id>"
  }
}
```

The `agent_did` is resolved from the Kubernetes pod annotation `moltrust.ch/agent-did`.

9.3 Falco Rule Design

`9.4 Sequential Action Safety (SAS)`

Falco rules are derived from AAE `mandate.deniedActions` entries. For each denied action pattern in the AAE, a corresponding Falco rule monitors the relevant syscalls.

Example: If an AAE contains `"deniedActions": ["/etc/*"]`, the Falco rule monitors `open`, `openat`, and `write` syscalls targeting `/etc/` paths for the annotated pod.

DID binding is established via Kubernetes pod annotations:

```
metadata:
  annotations:
    moltrust.ch/agent-did: "did:moltrust:<id>"
    moltrust.ch/aae-id: "urn:uuid:<aae-id>"
```

Status: Live. Test DID did:moltrust:662a7181e0154998 with seeded baseline 70.0. Falco Bridge Pod deployed on OpenClaw gateway. Reference implementation: https://github.com/HaraldeRoessler/moltrust-falco-bridge

---

9.5 MoltGraph – Interaction History Graph

Status: Live – Phase 2 **Deployed:** April 2026

Overview

MolTrust Trust Score answers: "How trustworthy is Agent B in general?" MoltGraph answers: "How has Agent B performed with agents that Agent A knows?"

Position in Stack

L2 AAE Authorization L2.5 SAS Pre-Execution L2.6 MoltGraph Query (Relationship-specific trust signal) L3 Trust Score Gate

Data Model

graph_edges table (live): from_did, to_did, context, outcome_score, source, interaction_at, on_chain_anchor.

Outcome mapping: CONFIRMED/grade_up=1.0, PARTIAL=0.6, grade_down=0.3, INCORRECT/revocation=0.0, INCONCLUSIVE=NULL.

Auto-Population Sources

- aeoess webhook grade_change/revocation: 1.0 / 0.0
- x402 USDC payments confirmed on Base: min(1.0, amount/5.0)
- WG cross-verification results: 1.0
- IPR verdicts (Phase 5): CONFIRMED=1.0, PARTIAL=0.6

Graph Query
2-hop neighbourhood with 45-day half-life decay and 50% hop-2 discount. Confidence = min(1.0, datapoints/10). graph_bonus activates only when confidence > 0.3.

API Endpoints (live)
GET /guard/api/graph/score/{from_did}/{to_did}?context= GET /guard/api/graph/neighbours/{did}?min_score= GET /guard/api/graph/stats POST /guard/api/graph/edge (API-key required)

Privacy: hop counts not exposed. All queries require API-key.

Trust Score Integration
GET /skill/trust-score/{did}?requester={requester_did} graph_bonus = neighbourhood_score x 10 x confidence (max +10 points)

Phases
Phase 1 (done): Table + seed edges + source column. Phase 2 (live): All endpoints + aeoess webhook + x402 hook. Phase 3 (next): graph_bonus in Trust Score formula.

10. Governance Layer

10.1 Layer Classification

The MolTrust protocol stack extends to four layers:

Layer	Name	Function	Status
1	Identity	W3C DID, Ed25519	Live
2	Authorization	AAE (Mandate/Constraints/Validity)	Live
3	Governance	Policy blocks, compliance rules	Planned Q3 2026
4	Execution	Runtime enforcement (API + Kernel)	API live, Kernel in progress

Layer 3 (Governance) sits between Authorization and Execution. It defines organizational and regulatory policies that constrain how AAEs are issued, what constraints are mandatory, and how violations are escalated.

The `VerifiedGovernanceCredential` VC type is planned for Q3 2026. It will encode organizational governance policies as machine-readable, signed artifacts.

The authority to activate runtime enforcement (advisory → enforce) is governed separately by the Combined Enforcement Protocol (Section 17.2); it is distinct from the organizational `VerifiedGovernanceCredential` policy artifact described here.

10.2 aps.txt Security Analysis

The Agent Provider Service (APS) protocol uses `aps.txt` files for provider discovery. A security analysis identified 5 attack vectors:

ID	Vector	Risk	Mitigation
AV-1	DNS Spoofing	aps.txt served from wrong domain	DNSSEC + HTTPS enforcement
AV-2	Content Manipulation	aps.txt modified in transit	TLS + signed aps.txt (proposed)
AV-3	Replay	Old aps.txt served after policy change	Timestamp + TTL in signed version
AV-4	DoS	aps.txt endpoint overwhelmed	CDN + rate limiting
AV-5	Confusion	Multiple aps.txt files with conflicting policies	Canonical resolution order

The strongest gap: `aps.txt` is currently unsigned. MolTrust has proposed signed `aps.txt` via DID-linked signatures as a Working Group contribution (Q2 2026). The bootstrapping rule for unknown authors is warn-not-block.

---

11. Outcome Verification

11.1 FlagRecord Schema

```
{
  "type": "FlagRecord",
  "marketId": "<market-id>",
  "flaggedAt": "<ISO-8601>",
  "anomalyScore": 55,
  "signals": ["volumeSpike", "priceVolumeDiv"],
  "assessment": "MEDIUM RISK",
  "source": "moltguard",
  "anchorBlock": 44098421
}
```

11.2 OutcomeRecord Schema

```
{
  "type": "OutcomeRecord",
  "marketId": "<market-id>",
  "resolvedAt": "<ISO-8601>",
  "outcome": "CONFIRMED | PARTIAL | INCONCLUSIVE | FALSE_POSITIVE",
  "flagRecordId": "<flag-record-id>",
  "settlementSource": "polymarket-gamma-api",
```

```
  "evidence": "<hash-of-settlement-data>"
}
```

11.3 FlagScore Formula

```
FlagScore = (CONFIRMED + 0.5 * PARTIAL + 0.25 * INCONCLUSIVE) / total_flags
```

FlagScore measures the accuracy of anomaly detection over time. A FlagScore of 1.0 means every flagged anomaly was subsequently confirmed. The score is computed per rolling quarter and anchored on-chain.

Note: The term "anomaly" is used deliberately rather than "manipulation" — flagged patterns indicate unusual activity that warrants investigation, not proven wrongdoing.

11.4 Settlement Watcher

The Settlement Watcher monitors the Polymarket Gamma API for market resolutions. When a previously flagged market resolves, it:

1. Creates an OutcomeRecord linking the flag to the resolution
2. Updates the running FlagScore
3. Triggers a Herald follow-up post if the outcome is CONFIRMED or PARTIAL

Status: Implementation ready, deployment in progress.

---

12. Agent Lifecycle (Layer A)

12.1 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).

12.2 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.

12.3 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.

12.4 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.

12.5 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.

12.6 Deregistration

On deregistration: DID marked inactive, credentials remain valid until their own expiry, behavioral record retained per Section 14.4, stake returned if no unresolved violations.

12.7 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.

---

13. Threat Model

13.1 Scope

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

13.2 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

13.3 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.

---

14. Privacy Model

14.1 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.

14.2 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

14.3 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.

14.4 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.

14.5 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.

---

15. 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.

---

16. Conformance

16.1 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

16.2 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

16.3 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.

16.4 Non-Goals

A conformant implementation is NOT required to:

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

16.5 Version Compatibility

Version 0.10 is a draft. Breaking changes to Layer A data formats will carry a minimum 12-month deprecation period in future versions. Layer B API changes follow semantic versioning. Layer C changes are non-breaking by definition.

---

17. Enforcement Layer & Governance Transition

This chapter describes (Part A) the AAE constraint-enforcement layer as **currently implemented in advisory mode**, and (Part B) the **designed** governance model that would authorize a transition to active enforcement. It is a roadmap-level overview: it states *what* the layer does and *what status* it has, not implementation internals.

Relationship to earlier chapters: this layer consumes the on-chain anchors of Section 6, sits alongside the runtime/kernel enforcement of Section 9 (Infrastructure-Layer Enforcement), and operates within the Governance Layer of Section 10. Section 9 enforces at the runtime/infrastructure boundary — cryptographic integrity (Layer 1), API-level trust-score and revocation consequences (Layer 2), kernel syscall detection (Layer 3 / Falco) and sequence safety (SAS). This chapter operates at the **credential/MANDATE-evaluation boundary**: whether a verified Agent Authorization Envelope permits a proposed action, producing signed verdicts. The two are complementary.

17.1 Part A — Enforcement Layer (implemented, advisory)

The AAE enforcement layer is **implemented and running in advisory mode**: it verifies authorization envelopes, evaluates proposed actions against them, and produces signed, auditable verdicts. In advisory mode a DENY verdict is **verified and recorded but not enforced** — the action is not blocked. Active blocking (the *enforce* mode) is a roadmap step gated on the governance transition of Part B (see §17.2.4).

The layer has three components.

17.1.1 Acceptance Gate (AAE verification at submit time)

Before an envelope is stored or evaluated, it is verified as a compact JWS (per the AAE schema, Section 2.8) — **fail-closed**:

- **Signature & signing-authority:** the envelope's signature is verified against the issuer's key, with an explicit `EdDSA`-only algorithm allow-list (no algorithm downgrade), and the signing DID MUST equal the credential issuer.
- **Schema:** the payload MUST carry a well-formed MANDATE / CONSTRAINTS / VALIDITY structure.
- **DID methods:** `did:moltrust` resolves against the reference registry (Section 5); `did:web` resolution is supported through an egress-controlled resolver.
- **Canonicalization:** verification binds to the exact signed bytes; the parsed structure is used only for schema checks.

An envelope that fails any check is rejected (not stored). A **default-DENY** rule governs evaluation: a required constraint that cannot be evaluated yields DENY, never a silent pass.

17.1.2 Evaluator (action evaluation, advisory)

The evaluator decides whether a proposed action is permitted by a verified envelope, evaluating the three AAE blocks:

- **MANDATE** — is the action within the granted scope;
- **CONSTRAINTS** — per-type checks (e.g. maximum transaction value, allowed domains, rate limits, single-use, validity window);
- **VALIDITY** — temporal validity (not-before / not-after, with clock-skew tolerance).

Each decision is recorded as a **signed verdict** (Ed25519, domain-separated, externally verifiable against the registry key) and, on DENY, an associated immutable violation record. Value-bearing constraints distinguish a **rail-verified** value from a merely client-asserted one; a client-asserted value on a required constraint yields DENY. This complements — and does not replace — the per-action gateway check referenced in Section 9.4 (MoltGuard): the registry-side evaluator produces a signed, anchorable verdict and violation record at the credential boundary.

Mode: the evaluator runs in **ADVISORY** mode. Violations are verified and logged (signed verdict + violation record, anchorable per Section 6); they are **not blocked**. The mandatory-chokepoint *enforce* mode — where a DENY prevents the action — is **Component 3 on the roadmap** and is gated per §17.2.4.

17.1.3 Envelope Store

Accepted envelopes are persisted in an append-only store with:

- a **content-hash primary key** (the key is the hash of the canonical signed bytes — content and identifier cannot diverge);
- a **single-use / replay guard** (a single-use envelope cannot be consumed twice for the same scope);
- **immutability** (no update or delete of a stored envelope; corrections are new envelopes).

This makes the evidence trail tamper-evident and consistent with the on-chain anchoring of Section 6.

17.2 Part B — Governance Transition (CEP, designed; not activated)

Part A runs in advisory mode today. Turning on active enforcement requires deciding **who is authorized** to flip the *enforce* switch. The Combined Enforcement Protocol (CEP) is the **designed** model for that authority. **CEP is not activated.** This section is a roadmap-level summary; the full design lives in the MolTrust ADR *CEP Governance* and the *CEP-3 Threshold Specification* (internal, accepted as design).

17.2.1 Problem

The authority to activate enforcement must not depend on a **person** (a founder does not survive a 10-year horizon), a **single chain** (it can disappear, censor, or fork), or a **single instance** (it can be shut down or compromised). CEP replaces such anchors with **objective, publicly recomputable conditions**.

17.2.2 Core concepts (overview)

- **5-condition ramp (AND).** The transition occurs only when five conditions hold *simultaneously*: a minimum elapsed time; a minimum number of Sybil-qualified relying parties;

distribution over a minimum number of independent clusters; no single actor above a voting-weight cap; and no single cluster above a share cap. A single threshold is manipulable — the AND-conjunction is not.

- **Honest-verifier data availability ("verification > production").** The condition data is published to decentralized, permanent storage and anchored as a `(merkle_root, data_uri)` tuple across multiple chains. The transition trigger is a **deterministic function anyone can recompute** from the published data — there is no privileged measuring party.
- **Keyed commitment + cryptographic erasure (data protection).** Relying-party identifiers are never anchored in clear; only keyed commitments are. Deletion destroys the key (cryptographic erasure), after which the anchored commitment is non-attributable — reconciling permanent integrity proofs with the right to erasure.
- **Staged verification.** A permissioned ramp-up phase (verifiers bound by data-processing agreements) precedes a target phase of zero-knowledge verification that proves the recomputation **without disclosing** the underlying data.
- **Scope position.** MolTrust is a **verification / enforcement protocol layer, not the deployer** of a high-risk AI system (cf. TLS/PKI: a certificate authority issues and lets parties verify; it does not supervise each use). The duty of human oversight over a high-risk *deployment* rests with the **relying party** that uses the protocol to gate its agents. The protocol is oversight-**enabling** (verifiable verdicts, default-DENY, audit trail, public veto), not oversight-replacing.

17.2.3 Testnet vs. Mainnet parameters (explicit)

The transition thresholds differ by network. Testnet values exist to **demonstrate the mechanism**; mainnet values are the **production target**. They are not interchangeable.

Parameter	Testnet (demonstration)	Mainnet (target)
N — Sybil-qualified relying parties	11	101
K — independent clusters	3	4
Y — max share per cluster	≥ 34 %	33 %
X — max voting weight per actor	10 %	10 %
T — timelock / public-veto window	31 days	31 days

Rationale for the K difference: with the byzantine-tolerance relation $N \geq 3f + 1$, **K = 4 (mainnet)** tolerates one captured/faulty cluster ($f = 1$) — true fault-tolerance hardening. **K = 3 (testnet)** corresponds to $f = 0$: it demonstrates the cluster-diversity mechanism end-to-end without claiming production-grade tolerance. The invariant $\mathbf{Y \geq 1/K}$ is preserved in both rows (testnet $K = 3 \longrightarrow Y \geq 34\ \% > 33.3\ \%$; mainnet $K = 4 \longrightarrow Y = 33\ \% \geq 25\ \%$), so neither configuration is structurally unsatisfiable.

The mechanism is therefore **demonstrable on testnet** while the mainnet configuration remains the (not-yet-activated) target.

17.2.4 Activation gates

Active *enforce* mode (Part A becoming blocking) is gated on two independent conditions, both of which must be met before any enforcement code is activated:

- **Gate-1 — thresholds fixed and anchored.** The parameters above are written down and chain-agnostically anchored before the ramp-up starts. *Status: the values are **fixed** (CEP-3 threshold specification); chain-agnostic anchoring is pending the multi-chain prerequisite below.*
- **Gate-2 — prerequisites built.** A relying-party registry with cryptographic (DID-bound) identity and cluster attribution; an explicit *enforce* state in the authorization model; and multi-chain anchoring. *Status: open.*

Until both gates are met, the evaluator remains in **advisory** mode (Part A): verified, recorded, **not blocked**. Activation is a deliberate, separate step, not a side effect of publishing this specification.

17.3 Summary of status

Element	Status
AAE acceptance gate (verify at submit)	Implemented
Evaluator (MANDATE/CONSTRAINTS/VALIDITY, signed verdicts)	Implemented — **advisory**
Envelope store (content-hash PK, replay guard, immutable)	Implemented
Active *enforce* mode (blocking)	Roadmap (gated, §17.2.4)
CEP governance transition	**Designed**, not activated
Mechanism on testnet	Demonstrable

This chapter adds no normative conformance requirement to Section 16; it documents an implemented advisory layer and a designed governance transition. Conformance for active enforcement will be specified when the activation gates (§17.2.4) are met.

---

18. Leak Canary

An agent that delegates work hands credentials to counterparties. When one of those credentials turns up where it was never meant to go, the delegating agent needs to learn two things: that it happened, and through which counterparty. A leak canary answers both. The delegating agent (the *issuer*) obtains one canary credential per counterparty; the reference registry records which counterparty received which canary and writes a signed leak event when the canary is presented by a party that should not hold it.

Relationship to earlier chapters: a canary is a credential of the reference registry, issued and verified like any other (Sections 2 and 3). The leak event carries a recomputable decision in the evidence format of the enforcement layer (Section 17.1), and a hold takes effect through the issuer's own mandate.

18.1 A canary is an ordinary credential

A canary is issued by the same issuance path as every other credential of the reference registry, with the same claims, keys, validity, storage, publication and anchoring (Section 6). No field, type, identifier pattern, validity period or status entry marks a canary. The mapping from a credential to its issuer and counterparty is held server-side by the registry and is readable only by the issuer.

The mechanism is public; the instances are not. A holder that knows how canaries work still cannot tell whether a given credential is one, and so has no reason to avoid presenting it. The reference implementation carries a public test that holds the structural equality of a canary and a regular credential from the same issuer.

18.2 Modes

The two modes match the two ways a delegating agent hands material out.

- **`bound`** — the credential's subject is the counterparty's DID, and the counterparty holds it as its own credential. Presentations by the issuer or by the counterparty are expected. A presentation by any other party is a leak.
- **`decoy`** — the credential's subject is also the counterparty's DID, but the counterparty receives the credential among other material and has no reason to present it. Every presentation except the issuer's own is a leak, including one by the counterparty.

The modes differ only in who counts as an expected presenter, not in the credential. In both modes the registry knows which counterparty received the canary, and a presenter that sends no API key counts as an unexpected presenter.

A canary can be issued only to a counterparty that is a registered, non-revoked agent of the reference registry. A request naming any other DID is refused before anything is issued or stored.

18.3 Flow

1. **Issue.** The issuer calls `POST /canary/issue` with the counterparty's DID, a mode and the reaction on leak (`hold` or `none`). The response carries the credential and a `canary_id`. A canary costs the same as any other credential.
2. **Hand over.** The issuer passes the credential to the counterparty through its usual channel. The registry is not involved in this step.
3. **Presentation.** A party presents the credential at `POST /credentials/verify`. The verifier's answer does not depend on whether the credential is a canary; the registry checks every presented credential against its canary records.
4. **Leak event.** If the credential is a canary and the presenter is not an expected holder (§18.2), the registry writes a signed leak event (§18.5). The presenter is identified only by the DID behind the API key it sent, if any.
5. **Hold.** For a canary issued with `on_leak: hold`, the counterparty's DID goes on the issuer's hold list.
6. **Enforce hook.** Before an enforcement check, the issuer passes its mandate through `POST /canary/enforce-hook`. The hook places, in front of every `allow` grant, a copy with disposition `hold` that applies only to held counterparties. The unchanged evaluator then returns `PENDING`

for those counterparties, which the issuer can ratify or let stand. The patched mandate is an ordinary mandate, and its verdicts recompute like any other.

7. **Review.** The issuer reads its canaries, events and holds (`GET /canary/list`, `GET /canary/events`, `GET /canary/holds`) and lifts a hold with `POST /canary/holds/release`. The same functions are available as MCP tools on the reference registry's MCP endpoint.

18.4 Attribution

A presentation proves that the credential has left the counterparty's hands. It ties the leak to the counterparty only when the counterparty's key signed the presentation. A presentation can carry an RFC 9421 HTTP Message Signature by the holder's DID over the request, in the proof-of-possession profile of the reference registry (covered components `@method`, `@path`, `x-moltrust-did`, `content-digest`; key in force from the registry; nonce valid once within 300 seconds). The signature covers the request body, and the body contains the credential.

The verifier checks such a signature on every presentation that carries one and reports the result in every answer, for every credential, in a `presentation` block (`api_key_did`, and `holder_proof` with `scheme`, `did`, `valid`, `reason`, `subject_match`). A holder signature that does not verify leaves the credential's own validity unchanged.

Each leak event records one of two attribution classes:

`attribution`	Condition	What the event shows
`counterparty`	Valid holder proof by the counterparty's DID	The counterparty's key signed this presentation of this credential
`open`	No holder proof, a proof that does not verify, or a proof by another DID	The leak; the path it took remains open

The class depends on the counterparty's key, not on the credential's subject, and applies the same way in both modes.

18.5 Leak event

The leak event is a JSON object with the members `event_version` (1), `event_id`, `canary_id`, `issuer_did`, `counterparty_did`, `mode`, `trigger` (`verifier_presentation`), `observed_at`, `presenter_did`, `credential_valid`, `canary_ref`, `presentation`, `attribution`, `aer` and `signature`.

- **`canary_ref`** is `sha256:` followed by the hex SHA-256 of the RFC 8785 (JCS) form of the credential the issuer received, so the issuer can match an event to a credential without asking the registry.
- **`presentation`** is the block from the verifier's answer. For a valid holder proof it adds a `record` with the public key, the signature, the signature base and the request body, so a third party can check the holder's signature and the bound path and body without contacting the registry.
- **`signature`** is an Ed25519 signature by the registry key over the bytes `moltrust:leak-canary-event:v1`, one `0x00` byte, and the JCS form of the event without the `signature` member. The public key is the registry JWK published in the issuer DID document; `GET /canary/trust-list` carries the same key.

- **`aer`** carries a complete, recomputable decision: one evidence item signed by `did:web:api.moltrust.ch#leak-canary` stating that the canary was presented, a one-grant mandate that allows the action `{"verb": "leak_canary_hold"}` only on that evidence, the evidence bundle and the decision record. The decision is `PERMIT`.

A third party checks an event offline with the `moltrust-enforce` package and the trust list: the event signature, the recomputed decision, the match between the signed evidence and the event's `canary_ref`, `counterparty_did` and `attribution`, and, for attribution `counterparty`, the holder proof. With the AER material, anyone can recompute that the hold follows from signed evidence.

18.6 Privacy

The registry stores DIDs, digests and timestamps for canaries and events. It stores no IP address, no user agent, no name, no e-mail address and no free text. The presenter appears only as the DID behind its API key, or as `null`. Every canary endpoint resolves the issuer from the API key and returns only that issuer's records; a request for another issuer's canary or event receives the same `404` as a request for one that does not exist. This version has no automatic expiry for canaries, events or holds; a hold ends when the issuer releases it. The general rules of Section 14 apply.

18.7 Limits

- **Attribution.** Credentials of the reference registry are published, so a party that obtains a published copy can present it. An event of class `open` may stem from such a copy and says nothing about the counterparty; only class `counterparty` names it.
- **Coverage.** A canary detects presentations at the reference verifier. A party that checks a leaked credential's signature on its own, or only stores the credential, leaves no trace.
- **Unknown presenters in `bound` mode.** A presenter without an API key counts as a leak. If the counterparty has its own relying parties verify the credential, those presentations raise events; `bound` canaries suit counterparties that are expected to present to the issuer alone.
- **No per-counterparty `forbid`.** The evaluator applies `forbid` to the action binding before any constraint is read, so a per-counterparty forbid would forbid the action for every counterparty. An issuer that wants this removes the counterparty from its `allow` grants instead.

18.8 Summary of status

Element	Status
Canary issuance, modes `bound` and `decoy`	Live
Detection at the reference verifier, signed leak event with AER material	Live
Holder-proof attribution (`counterparty` / `open`)	Live
Hold list and enforce hook (`PENDING` for held counterparties)	Live
Per-counterparty `forbid`	Not offered (§18.7)

This chapter adds no normative conformance requirement to Section 16; it documents a live service of the reference registry.

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
- 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

---

This document is released under Creative Commons Attribution 4.0 International (CC BY 4.0). The protocol is open. The reference implementation is operated by MolTrust.