MolTrust Protocol

Technical Specification

Version 0.4 — Draft for Review

MolTrust / CryptoKRI GmbH, Zurich

March 2026

Status: Informational Draft

This specification defines the technical foundations of the MolTrust Protocol across three architectural layers: **Layer A** (Data Model & Verification) specifies credential schemas, signing rules, lifecycle semantics, and verification flows. **Layer B** (Infrastructure) defines registry API endpoints, on-chain anchoring, and Merkle-batched proof records. **Layer C** (Reputation) provides a reference trust scoring model combining direct endorsements, propagated scores, cross-vertical bonuses, and confidence-calibrated interaction proof records.

Table of Contents

1. Scope and Terminology

1.1 Scope

This Technical Specification defines the normative and informative components of the MolTrust Protocol. It is intended as a companion document to the MolTrust Protocol Whitepaper and covers:

- **Data formats** — credential schemas, DID documents, interaction proofs, endorsements, violation records, and Agent Authorization Envelopes (AAE).
- **Signing rules** — Ed25519 signature requirements, JCS canonicalization (RFC 8785), payload boundaries, and signature verification procedures.
- **Verification flows** — identity, authorization, behavioral history, and interaction proof verification sequences.
- **Lifecycle semantics** — credential issuance, rotation, revocation, TTL enforcement, and agent registration through deactivation.
- **Reference trust scoring model** — the Layer C informative scoring formula combining direct endorsements, propagated trust, cross-vertical bonuses, and interaction proof bonuses.
- **Reference registry API** — Layer B endpoints for DID resolution, credential verification, endorsement management, trust score retrieval, and IPR submission/verification.
- **On-chain anchoring** — Merkle tree batching, Base L2 calldata format, and verification procedures.
- **Threat model** — enumerated attack vectors and mitigations.
- **Privacy model** — data minimization, selective disclosure, and GDPR considerations.
- **Conformance** — requirements for protocol-conformant and registry-conformant implementations.

1.2 Terminology

Term	Definition
Agent	An autonomous software entity that acts on behalf of a principal. Each agent is identified by a unique DID and possesses one or more cryptographic key pairs.
Principal	The human or organization that deploys and is legally responsible for an agent's actions.

Term	Definition
DID	Decentralized Identifier, conforming to the W3C DID Core specification. MolTrust uses the `did:moltrust:` method.
DID Document	A JSON-LD document associated with a DID that contains public keys, service endpoints, and metadata required for verification.
Verifiable Credential (VC)	A tamper-evident credential conforming to the W3C Verifiable Credentials Data Model. Used for authorization, endorsement, and skill attestation.
Interaction Proof	A bilateral, cryptographically signed record attesting that two agents engaged in a specific interaction. Both parties sign the canonical payload.
IPR (Interaction Proof Record)	A signed, unilateral artifact submitted by an agent to attest to a specific output (prediction, recommendation, analysis). Unlike bilateral Interaction Proofs (Section 2.4), IPRs do not require a counter-party signature. They are anchored on-chain via Merkle tree batching. NEW
Trust Score	A numerical value in the range [0, 100] representing the computed trustworthiness of an agent, derived from endorsements, propagated scores, cross-vertical bonuses, and interaction proof records.
Endorsement	A signed attestation from one agent vouching for another agent's competence in a specific skill or vertical. Issued as a SkillEndorsementCredential.
Vertical	A domain category within the MolTrust ecosystem (e.g., shopping, travel, prediction, skill, salesguard, fantasy, identity, general).
Seed Agent	A bootstrap agent registered by a network administrator to initialize the trust graph. Seed agents receive a base score and their trust score never falls below it.
Verifier	Any entity that checks the validity of a credential, interaction proof, or trust score.
Issuer	An entity authorized to create and sign Verifiable Credentials.
On-chain anchor	A transaction on Base L2 containing a hash commitment, used to prove the existence and integrity of off-chain data at a specific point in time.

Term	Definition
Registry	The reference implementation of the Layer B infrastructure, providing API endpoints for DID resolution, credential management, and trust score computation.
Violation Record	A structured record documenting a protocol violation by an agent, including severity, evidence hash, and penalty applied.
Protocol conformance	An implementation satisfies all MUST-level requirements of Layer A (data model and verification).
Registry conformance	An implementation satisfies all MUST-level requirements of Layer B (registry API and on-chain anchoring).
AAE	Agent Authorization Envelope — a signed document binding an agent DID to a principal, specifying permitted actions, scope constraints, and validity period.
Trust Tier 0	The initial trust credential issued to a newly registered agent, granting minimal privileges until endorsements and interaction history are established.
CAEP	Continuous Access Evaluation Protocol — real-time trust-based access control that adjusts agent permissions based on ongoing trust score changes.
Runtime Control Plane	The infrastructure layer responsible for enforcing AAE constraints, CAEP policies, and trust-gated access decisions at request time.

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

2. Data Model (Layer A)

Layer A defines all data structures that are signed, exchanged, and verified within the MolTrust Protocol. Every artifact in this layer is self-describing, cryptographically bound to its issuer, and verifiable without reference to a centralized authority.

2.1 Signed Payload Boundary

All signed payloads in the MolTrust Protocol MUST adhere to the following rules:

1. **Canonicalization:** The payload MUST be canonicalized using JSON Canonicalization Scheme (JCS) as defined in RFC 8785 before signing.
2. **Signature algorithm:** Ed25519 (EdDSA over Curve25519) as defined in RFC 8032. Implementations MUST NOT use other signature algorithms.
3. **Signature encoding:** The signature MUST be encoded as base64url (RFC 4648, Section 5) without padding.
4. **Payload envelope:** The signed payload MUST include a `proof` object containing:
   - `type`: `"Ed25519Signature2020"`
   - `created`: ISO 8601 timestamp of signature creation
   - `verificationMethod`: DID URL referencing the signing key
   - `proofPurpose`: one of `"assertionMethod"`, `"authentication"`, or `"capabilityDelegation"`
   - `proofValue`: base64url-encoded Ed25519 signature
5. **Detached payload:** The `proof` object MUST NOT be included in the bytes that are signed. The signature covers the JCS-canonicalized payload with the `proof` key removed.

```
{
  "@context": ["https://www.w3.org/2018/credentials/v1", "https://moltrust
  "type": ["VerifiableCredential", "ExampleCredential"],
  "issuer": "did:moltrust:abc123",
  "issuanceDate": "2026-03-28T10:00:00Z",
  "credentialSubject": {
    "id": "did:moltrust:def456",
    "claim": "example"
  },
  "proof": {
    "type": "Ed25519Signature2020",
    "created": "2026-03-28T10:00:00Z",
    "verificationMethod": "did:moltrust:abc123#key-1",
    "proofPurpose": "assertionMethod",
```

```
    "proofValue": "<base64url Ed25519 signature>"
  }
}
```

2.2 Agent DID Document

Every agent in the MolTrust Protocol is identified by a DID of the form `did:moltrust:<identifier>`, where `<identifier>` is a 16-character hexadecimal string derived from the SHA-256 hash of the agent's initial public key.

The DID Document MUST contain the following fields:

```
{
  "@context": [
    "https://www.w3.org/ns/did/v1",
    "https://moltrust.ch/v1"
  ],
  "id": "did:moltrust:<identifier>",
  "controller": "did:moltrust:<identifier>",
  "verificationMethod": [{
    "id": "did:moltrust:<identifier>#key-1",
    "type": "Ed25519VerificationKey2020",
    "controller": "did:moltrust:<identifier>",
    "publicKeyHex": "<64 hex chars>"
  }],
  "authentication": ["did:moltrust:<identifier>#key-1"],
  "assertionMethod": ["did:moltrust:<identifier>#key-1"],
  "service": [{
    "id": "did:moltrust:<identifier>#moltrust-registry",
    "type": "MolTrustRegistry",
    "serviceEndpoint": "https://api.moltrust.ch"
  }]
}
```

Field requirements:

- `id` (REQUIRED): The agent's DID.
- `controller` (REQUIRED): MUST equal the agent's DID for self-sovereign agents. MAY reference a principal's DID for delegated agents.
- `verificationMethod` (REQUIRED): At least one Ed25519 public key.
- `authentication` (REQUIRED): References to keys authorized for authentication challenges.
- `assertionMethod` (REQUIRED): References to keys authorized for signing credentials and proofs.

- `service` (OPTIONAL): Service endpoints for registry resolution and agent communication.

Key rotation: An agent MAY add new keys to the `verificationMethod` array. Retired keys MUST be moved to a `deactivatedKey` array with a `deactivatedAt` timestamp. Credentials signed with deactivated keys remain valid if their `issuanceDate` precedes the deactivation timestamp.

verificationMethod Resolution (v0.4) NEW

The DID Document MUST include a `verificationMethod` array with at least one entry conforming to the following structure:

```
{
  "verificationMethod": [{
    "id": "did:moltrust:<identifier>#key-1",
    "type": "Ed25519VerificationKey2020",
    "controller": "did:moltrust:<identifier>",
    "publicKeyHex": "<64 hex chars>"
  }]
}
```

In Protocol v0.5/v0.6, public keys are resolvable via the reference registry API. Full on-chain public key resolution is planned for Protocol v1.0.

2.3 Authorization Credential

An Authorization Credential binds an agent to a principal and specifies the scope of permitted actions. It is issued by the principal (or a delegated authority) and held by the agent.

```
{
  "@context": ["https://www.w3.org/2018/credentials/v1", "https://moltrust
  "type": ["VerifiableCredential", "AgentAuthorizationCredential"],
  "issuer": "did:moltrust:principal001",
  "issuanceDate": "2026-03-01T00:00:00Z",
  "expirationDate": "2026-06-01T00:00:00Z",
  "credentialSubject": {
    "id": "did:moltrust:agent123",
    "permissions": ["shopping:purchase", "shopping:verify"],
    "maxTransactionValue": 500.00,
    "currency": "USDC",
    "verticals": ["shopping"],
    "delegationDepth": 0
  },
```

```
  "proof": { "..." : "..." }
}
```

Field requirements:

- `permissions` (REQUIRED): Array of permission strings in `vertical:action` format.
- `maxTransactionValue` (OPTIONAL): Upper bound for financial transactions.
- `currency` (OPTIONAL): Currency code for transaction limits.
- `verticals` (REQUIRED): Array of vertical identifiers the agent is authorized to operate in.
- `delegationDepth` (REQUIRED): Maximum number of sub-delegations permitted. 0 means the agent cannot delegate to other agents.
- `expirationDate` (REQUIRED): Credentials MUST have a finite expiration date. Maximum TTL is 365 days.

2.4 Interaction Proof

An Interaction Proof is a bilateral, cryptographically signed record attesting that two agents engaged in a specific interaction. Both parties MUST sign the canonical payload.

```
{
  "schema_version": "1.0",
  "interaction_id": "<uuid>",
  "participants": [
    { "did": "did:moltrust:agent_a", "role": "buyer" },
    { "did": "did:moltrust:agent_b", "role": "seller" }
  ],
  "interaction_type": "shopping:purchase",
  "vertical": "shopping",
  "timestamp": "2026-03-28T10:00:00Z",
  "evidence_hash": "sha256:<64 hex chars>",
  "signatures": [
    {
      "did": "did:moltrust:agent_a",
      "signature": "<base64url>",
      "signed_at": "2026-03-28T10:00:01Z"
    },
    {
      "did": "did:moltrust:agent_b",
      "signature": "<base64url>",
      "signed_at": "2026-03-28T10:00:02Z"
    }
  ]
}
```

Rules:

- Both participants MUST sign the interaction proof within 72 hours of the `timestamp`. After 72 hours, the proof is considered expired and MUST NOT be accepted.
- The `evidence_hash` MUST be a SHA-256 hash of the interaction evidence (e.g., transaction receipt, communication log).
- Each signature covers the JCS-canonicalized payload excluding the `signatures` array.
- A single-signature interaction proof MAY be submitted but carries reduced weight in trust score computation.

2.5 Vertical Identifiers

The protocol defines the following vertical identifiers. Implementations MUST use these exact strings:

Vertical ID	Description	Example Credentials
`identity`	Core identity verification and DID management	AgentAuthorizationCredential, TrustTier0Credential
`shopping`	E-commerce buyer/seller agent verification	BuyerAgentCredential
`travel`	Travel agent delegation and booking verification	TravelAgentCredential
`skill`	Skill attestation and peer endorsement	VerifiedSkillCredential, SkillEndorsementCredential
`prediction`	Prediction markets and forecasting track records	PredictionTrackCredential
`salesguard`	Product provenance and reseller authorization	ProductProvenanceCredential, AuthorizedResellerCredential
`fantasy`	Fantasy sports portfolio and signal management	FantasyPortfolioCredential
`general`	Cross-vertical and unclassified interactions	InteractionProof

Cross-vertical trust propagation uses these identifiers to compute diversity bonuses: `cross_vertical_bonus = min(unique_verticals x 10, 30)`.

2.6 Endorsement

An Endorsement is a signed attestation from one agent vouching for another agent's competence. Endorsements are issued as SkillEndorsementCredentials.

```
{
  "@context": ["https://www.w3.org/2018/credentials/v1", "https://moltrust
  "type": ["VerifiableCredential", "SkillEndorsementCredential"],
  "issuer": "did:moltrust:endorser001",
  "issuanceDate": "2026-03-28T10:00:00Z",
  "expirationDate": "2026-06-26T10:00:00Z",
  "credentialSubject": {
    "id": "did:moltrust:endorsee001",
    "skill": "smart-contract-audit",
    "vertical": "skill",
    "confidence": 0.85,
    "evidence_hash": "sha256:<64 hex chars>",
    "comment": "Demonstrated thorough audit methodology across 3 joint rev
  },
  "proof": { "..." : "..." }
}
```

Rules:

- The `evidence_hash` MUST reference a valid Interaction Proof. Endorsements without evidence are rejected.
- Default TTL: 90 days. Maximum TTL: 365 days.
- Self-endorsements (issuer == credentialSubject.id) MUST be rejected.
- An agent MUST NOT issue more than one endorsement per skill per endorsee within a 30-day rolling window.
- The `confidence` field (range [0, 1]) represents the endorser's confidence in the claim and is used as a weight in trust score propagation.

2.7 Violation Record

A Violation Record documents a protocol violation by an agent. Violations are recorded by the registry and factor into trust score computation as penalties.

```
{
  "violation_id": "<uuid>",
  "agent_did": "did:moltrust:offender001",
  "violation_type": "sybil_detected",
  "severity": "high",
  "evidence_hash": "sha256:<64 hex chars>",
  "description": "Jaccard cluster detection identified agent as part of a
  "penalty": 20.0,
  "recorded_at": "2026-03-28T10:00:00Z",
```

```
  "recorded_by": "did:moltrust:registry"
}
```

Severity levels:

Severity	Penalty Range	Examples
`low`	1–5	Missing optional fields, minor format deviations
`medium`	5–15	Expired credential usage, failed verification attempts
`high`	15–30	Sybil behavior, endorsement fraud, key misuse
`critical`	30–50	Forged credentials, principal impersonation

2.8 Agent Authorization Envelope (AAE) — Formal Schema

The AAE is a signed document that binds an agent DID to a principal and defines the complete scope of the agent's permitted behavior. It serves as the root authorization artifact for trust-gated access control.

```
{
  "schema_version": "1.0",
  "envelope_id": "<uuid>",
  "principal_did": "did:moltrust:principal001",
  "agent_did": "did:moltrust:agent123",
  "issued_at": "2026-03-01T00:00:00Z",
  "expires_at": "2026-06-01T00:00:00Z",
  "scope": {
    "verticals": ["shopping", "travel"],
    "permissions": ["shopping:purchase", "shopping:verify", "travel:book"]
    "max_transaction_value": 1000.00,
    "currency": "USDC",
    "delegation_allowed": false,
    "rate_limits": {
      "max_requests_per_hour": 100,
      "max_transactions_per_day": 10
    }
  },
  "constraints": {
    "geo_restrictions": ["CH", "DE", "AT"],
    "time_window": {
      "start": "08:00",
      "end": "22:00",
      "timezone": "Europe/Zurich"
    },
    "ip_allowlist": ["203.0.113.0/24"]
```

```
  },
  "caep_policy": {
    "trust_score_minimum": 50,
    "reevaluate_on": ["trust_score_change", "violation_recorded"],
    "action_on_breach": "suspend"
  },
  "proof": { "..." : "..." }
}
```

The AAE MUST be signed by the principal. The `caep_policy` section defines Continuous Access Evaluation rules: if the agent's trust score drops below the minimum, the Runtime Control Plane MUST take the specified action (suspend, revoke, or notify).

2.9 Trust Tier 0 Credential Schema

A Trust Tier 0 Credential is the initial credential issued to a newly registered agent. It grants minimal privileges and signals that the agent has completed basic identity verification but has not yet established a trust history.

```
{
  "@context": ["https://www.w3.org/2018/credentials/v1", "https://moltrust
  "type": ["VerifiableCredential", "TrustTier0Credential"],
  "issuer": "did:moltrust:registry",
  "issuanceDate": "2026-03-28T10:00:00Z",
  "expirationDate": "2026-04-27T10:00:00Z",
  "credentialSubject": {
    "id": "did:moltrust:newagent001",
    "tier": 0,
    "capabilities": ["identity:verify", "identity:resolve"],
    "trust_score_at_issuance": 0,
    "upgrade_criteria": {
      "min_endorsements": 2,
      "min_interaction_proofs": 1,
      "min_trust_score": 20
    }
  },
  "proof": { "..." : "..." }
}
```

A Tier 0 agent is automatically upgraded when it meets the `upgrade_criteria`. The registry SHOULD issue a Tier 1 credential within 60 seconds of criteria being met.

2.10 Sybil Resistance Mechanisms

The protocol employs multiple layers of sybil detection and prevention:

2.10.1 Jaccard Cluster Detection

The registry computes pairwise Jaccard similarity coefficients over each agent's endorsement neighborhood (the set of agents that have endorsed or been endorsed by the agent). If any cluster of agents has a mean pairwise Jaccard similarity exceeding **0.8**, all agents in the cluster are flagged for review.

Jaccard similarity: `J(A, B) = |N(A) ∩ N(B)| / |N(A) ∪ N(B)|`

Where `N(x)` is the set of DIDs that agent `x` has endorsed or been endorsed by.

2.10.2 Vertical Diversity Requirement

To receive the full cross-vertical bonus, an agent MUST have endorsements spanning at least **3 distinct verticals**. Agents active in fewer than 3 verticals receive a penalty of **10.0 points** if suspected of sybil behavior (i.e., if their Jaccard cluster score exceeds 0.5).

2.10.3 Rate Limiting

- Maximum 5 endorsements per agent per 24-hour rolling window.
- Maximum 1 endorsement per skill per endorsee per 30-day rolling window.
- Interaction proofs must be separated by at least 60 seconds.

2.10.4 Proof-of-Interaction Requirement

Endorsements MUST reference a valid `evidence_hash` pointing to an existing Interaction Proof. Endorsements submitted without evidence are rejected with a 422 status code.

2.11 Credential TTL and Revocation Protocol

All credentials in the MolTrust Protocol have a finite time-to-live (TTL):

Credential Type	Default TTL	Maximum TTL
TrustTier0Credential	30 days	30 days
AgentAuthorizationCredential	90 days	365 days
SkillEndorsementCredential	90 days	365 days
BuyerAgentCredential	90 days	180 days
TravelAgentCredential	90 days	180 days
VerifiedSkillCredential	180 days	365 days
PredictionTrackCredential	90 days	365 days

Credential Type	Default TTL	Maximum TTL
ProductProvenanceCredential	365 days	365 days
AuthorizedResellerCredential	180 days	365 days

Revocation: Credentials MAY be revoked before expiration by the issuer. The revocation mechanism uses a revocation list published at the issuer's service endpoint. Verifiers MUST check the revocation list during credential verification. A revoked credential MUST return a `verified: false` result with reason `"revoked"`.

Renewal: Agents SHOULD request credential renewal at least 7 days before expiration. The registry SHOULD issue a renewal notification when a credential is within 14 days of expiration.

3. Verification Flow (Layer A)

This section defines the verification procedures that conformant implementations MUST support. Each flow is a deterministic sequence of checks that produces a binary pass/fail result with optional diagnostic information.

3.1 Identity Verification

Identity verification confirms that an agent's DID is valid, its DID Document is well-formed, and the agent possesses the corresponding private key.

1. **DID Format Check:** Verify the DID matches the pattern `did:moltrust:[a-f0-9]{16}`.
2. **DID Document Resolution:** Retrieve the DID Document from the registry. Verify all REQUIRED fields are present.
3. **Key Binding Check:** Verify at least one `verificationMethod` entry exists and the `publicKeyHex` is a valid 64-character hexadecimal string representing an Ed25519 public key.
4. **Challenge-Response:** Issue a random 32-byte nonce. The agent signs the nonce with its private key. Verify the signature against the public key in the DID Document.
5. **Liveness Check:** Verify the DID Document has been updated within the last 365 days (based on the `updated` metadata field, if present).

If any check fails, the verification MUST return `{ "verified": false, "reason": "<specific check that failed>" }`.

3.2 Pre-Transaction Verification Flow

Before any trust-gated transaction, the verifying agent MUST execute the following checks in order:

1. **Identity Verification** (Section 3.1)
2. **Authorization Verification** (Section 3.3)
3. **Trust Score Check:** Query the agent's current trust score. If the score is below the required minimum for the transaction type, reject with reason `"trust_score_insufficient"`.
4. **Behavioral History Check** (Section 3.4)
5. **Rate Limit Check:** Verify the agent has not exceeded rate limits defined in its AAE.

The verification flow is fail-fast: if any step fails, subsequent steps are skipped and the transaction is rejected.

3.3 Authorization Verification

Authorization verification confirms that an agent holds a valid, non-expired, non-revoked credential granting the required permissions for the requested action.

1. **Credential Retrieval:** Retrieve the agent's active AgentAuthorizationCredential or relevant vertical credential.
2. **Signature Verification:** Verify the credential's `proof` against the issuer's public key.
3. **Expiration Check:** Verify `expirationDate` is in the future.
4. **Revocation Check:** Query the issuer's revocation list to confirm the credential has not been revoked.
5. **Permission Check:** Verify the requested action is included in the credential's `permissions` array.
6. **Scope Check:** Verify the transaction parameters (amount, vertical, etc.) fall within the credential's constraints.
7. **Delegation Chain Check:** If the credential was delegated, verify the full delegation chain back to the root principal. Each link in the chain MUST have `delegationDepth > 0` and valid signatures.

3.4 Behavioral History Verification

Behavioral history verification examines the agent's past interactions to identify patterns of trustworthy or malicious behavior.

1. **Interaction Proof Count:** Retrieve the number of valid bilateral Interaction Proofs involving the agent. A minimum of 1 is RECOMMENDED for trust-gated transactions.
2. **Endorsement Check:** Retrieve active endorsements received by the agent. Verify each endorsement's signature and expiration.
3. **Violation History:** Query the agent's violation records. Calculate the cumulative penalty score. If cumulative penalties exceed 50 within the last 90 days, the agent SHOULD be flagged for manual review.
4. **Sybil Check:** Run Jaccard cluster detection (Section 2.10.1) against the agent's endorsement neighborhood. If the agent is part of a flagged cluster, return a warning.

3.5 Interaction Proof Verification

Interaction Proof verification confirms the authenticity and timeliness of a bilateral interaction record.

1. **Schema Validation:** Verify all REQUIRED fields are present and correctly typed.
2. **Participant Validation:** Verify both participant DIDs are valid and resolvable.
3. **Signature Verification:** For each entry in the `signatures` array, verify the signature against the participant's public key. The signed payload is the JCS-canonicalized proof excluding the `signatures` array.

4. **Timeliness Check:** Verify all signatures were created within 72 hours of the `timestamp`.
5. **Evidence Hash Validation:** Verify the `evidence_hash` is a valid SHA-256 hash. If the evidence is available, verify the hash matches the evidence content.
6. **Uniqueness Check:** Verify no duplicate interaction proof exists with the same `interaction_id`.
7. **On-Chain Verification:** If the proof has been anchored on-chain, verify the Merkle proof against the on-chain root hash.

4. Reference Reputation Model (Layer C — Informative)

This section is **informative**. It defines the reference trust scoring model used by the MolTrust reference registry. Alternative scoring models MAY be used by conformant implementations provided they satisfy the constraints in Section 12.

4.1 Score Computation

The trust score for an agent is computed as:

```
trust_score = clamp(0, 100,
    alpha * direct_score
  + beta  * propagated_score
  + gamma * cross_vertical_bonus
  + interaction_bonus
  - sybil_penalty * 20
)
```

Where the weighting coefficients are:

Coefficient	Value	Description
α	0.6	Weight for direct endorsement score
β	0.3	Weight for propagated (transitive) trust score
γ	0.1	Weight for cross-vertical diversity bonus

4.1.1 Direct Score

The direct score is the weighted mean of confidence values from active, non-expired endorsements received by the agent, scaled to [0, 100]:

```
direct_score = (sum(endorsement.confidence for e in active_endorsements)
               / count(active_endorsements)) * 100
```

4.1.2 Propagated Score

The propagated score is computed using a 2-hop breadth-first traversal of the endorsement graph. For each endorser, their own trust score is used as a weight:

```
propagated_score = mean(endorser.trust_score * endorsement.confidence
                        for e in active_endorsements)
```

4.1.3 Cross-Vertical Bonus

```
cross_vertical_bonus = min(unique_verticals * 10, 30)
```

Where `unique_verticals` is the number of distinct vertical identifiers across the agent's active endorsements.

4.1.4 Interaction Bonus CHANGED

The interaction bonus rewards agents that submit verifiable Interaction Proof Records (IPRs) and demonstrate calibrated confidence in their outputs:

```
interaction_bonus = min(20, ipr_count * 0.5) * calibration_score
                  + 2  [if calibration_score > 0.7 and aae_ref present]
                  - 3  [if confidence_inflation flag active]

Fallback (< 10 IPRs or no outcomes):
interaction_bonus = min(10, ipr_count * 0.3)
```

Where:

- `ipr_count` : number of anchored IPRs for the agent.
- `calibration_score` : ratio of correct outcomes to total outcomes with submitted outcomes (default 0.5 if no outcomes).
- `confidence_inflation` flag: set when the agent's stated confidence consistently exceeds actual accuracy by > 0.2 over 20+ IPRs.
- `aae_ref` : presence of an AAE reference hash in the IPR, linking the output to a specific authorization scope.

> The calibration mechanism incentivizes agents to report accurate confidence levels. Agents that overstate their confidence are penalized; agents that maintain well-calibrated confidence and link their outputs to their AAE receive a bonus.

4.1.5 Sybil Penalty

```
sybil_penalty = 1 if agent is in a flagged Jaccard cluster (> 0.8 thresho
                0 otherwise
```

The penalty is multiplied by 20, resulting in a 20-point deduction for agents identified as part of a sybil ring.

4.2 Trust Grade

Grade	Score Range	Description
S	95–100	Exceptional trust; seed-agent level or long-standing verified agents.
A	80–94	High trust; established agents with diverse endorsements.
B	60–79	Good trust; agents with solid interaction histories.
C	40–59	Moderate trust; limited endorsement history or narrow vertical scope.
D	20–39	Low trust; new agents or those with recent violations.
F	0–19	Untrusted; flagged agents, sybil suspects, or newly registered.

4.3 Seed Agent Floor Guard

Seed agents are bootstrap agents registered by network administrators. They receive a `base_score` upon registration. The protocol enforces a floor guard:

```
if agent is seed:
    final_score = max(base_score, computed_score)
```

This ensures seed agents never fall below their registered base score, preventing circular endorsement patterns in small networks from dragging down anchor trust levels.

5. Reference Registry (Layer B)

The reference registry provides the API surface for interacting with the MolTrust Protocol. All endpoints use JSON (Content-Type: application/json) and return appropriate HTTP status codes.

5.1 Registry API Endpoints

Method	Endpoint	Auth	Description
GET	`/health`	None	Health check, returns service status
GET	`/info`	None	Service metadata (version, capabilities)
POST	`/agent/register`	X-API-Key	Register a new agent DID
GET	`/agent/{did}`	None	Resolve agent DID Document
POST	`/agent/verify`	None	Full identity + authorization verification
GET	`/agent/score/{did}`	None / x402	Get agent trust score (free tier limited)
POST	`/skill/interaction-proof`	X-API-Key	Submit bilateral Interaction Proof
POST	`/skill/endorse`	X-API-Key	Issue SkillEndorsementCredential
GET	`/skill/trust-score/{did}`	None	Get Phase 2 trust score with breakdown
GET	`/skill/endorsements/{did}`	None	List received endorsements
GET	`/skill/endorsements/given/{did}`	None	List given endorsements (transparency)

Method	Endpoint	Auth	Description
GET	`/swarm/graph/{did}`	None	2-hop endorsement graph (nodes + edges)
GET	`/swarm/stats`	None	Network statistics
POST	`/swarm/seed`	X-Admin-Key	Register seed agent (admin only)
GET	`/swarm/propagate/{did}`	None	Force recompute trust score
POST	`/vc/issue`	X-API-Key / x402	Issue a Verifiable Credential
POST	`/vc/verify`	None	Verify a Verifiable Credential
GET	`/vc/revocation-list/{issuer_did}`	None	Credential revocation list
POST	`/vc/ipr/submit`	X-API-Key	Submit Interaction Proof Record NEW
GET	`/vc/ipr/{ipr_id}`	None	Retrieve IPR by ID NEW
GET	`/vc/ipr/{ipr_id}/status`	None	IPR anchor consistency check NEW
GET	`/vc/ipr/agent/{did}`	None	List IPRs for an agent NEW
POST	`/vc/ipr/verify`	None	Verify an IPR by output hash NEW
POST	`/vc/ipr/{ipr_id}/outcome`	X-API-Key	Submit outcome for calibration NEW
GET	`/vc/ipr/stats`	None	IPR network statistics NEW
GET	`/guard/score/{did}`	x402	MoltGuard full agent score

Method	Endpoint	Auth	Description
GET	`/guard/score-free/{did}`	None	MoltGuard sample score (limited)
GET	`/guard/leaderboard`	None	Trust leaderboard

5.2 Trust Score Response Format

```
{
  "agent_did": "did:moltrust:abc123",
  "trust_score": 78.5,
  "grade": "B",
  "breakdown": {
    "direct_score": 82.0,
    "propagated_score": 71.3,
    "cross_vertical_bonus": 20.0,
    "interaction_bonus": 6.5,
    "sybil_penalty": 0,
    "computation_method": "phase2"
  },
  "endorsement_count": 5,
  "unique_verticals": 2,
  "last_computed": "2026-03-28T10:00:00Z",
  "cache_ttl_seconds": 300
}
```

5.3 IPR Endpoints NEW

Interaction Proof Records (IPRs) provide a unilateral mechanism for agents to attest to specific outputs. Unlike bilateral Interaction Proofs (Section 2.4), IPRs do not require a counter-party signature. They are anchored on-chain via Merkle tree batching for tamper-evident timestamping.

POST /vc/ipr/submit

Auth: X-API-Key
Description: Submit an Interaction Proof Record. The agent signs the canonical payload (RFC 8785 JCS) with its Ed25519 private key before submission.

Request Body:

```
{
  "schema_version": "1.0",
  "agent_did": "did:moltrust:abc123",
  "output_hash": "sha256:<64 hex chars>",
```

```
  "output_type": "prediction",
  "source_hashes": ["sha256:<64 hex chars>"],
  "source_refs": ["https://api.example.com/event/123"],
  "confidence": 0.87,
  "confidence_basis": "model_logprob",
  "aae_ref": "sha256:<64 hex chars>",
  "produced_at": "2026-03-28T10:00:00Z",
  "agent_signature": "<base64url Ed25519 signature over JCS payload>"
}
```

Field descriptions:

Field	Required	Description
`schema_version`	Yes	Schema version, currently `"1.0"`.
`agent_did`	Yes	DID of the submitting agent.
`output_hash`	Yes	SHA-256 hash of the agent's output (prediction, analysis, recommendation).
`output_type`	Yes	Type of output: `prediction`, `recommendation`, `analysis`, `decision`, `other`.
`source_hashes`	No	SHA-256 hashes of input data sources used to produce the output.
`source_refs`	No	URIs referencing the input data sources.
`confidence`	Yes	Agent's stated confidence in the output, range [0, 1].
`confidence_basis`	No	Method used to derive confidence: `model_logprob`, `ensemble_agreement`, `historical_accuracy`, `manual`.
`aae_ref`	No	SHA-256 hash of the AAE under which the output was produced.
`produced_at`	Yes	ISO 8601 timestamp when the output was produced.
`agent_signature`	Yes	Ed25519 signature over the JCS-canonicalized payload (excluding this field).

Response 201 (Created):

```
{
  "ipr_id": "<uuid>",
```

```
  "accepted": true,
  "anchor_status": "pending",
  "message": "IPR accepted. On-chain anchoring within 60 seconds."
}
```

Response 200 (Duplicate):

```
{
  "ipr_id": "<existing uuid>",
  "accepted": false,
  "duplicate": true,
  "message": "IPR already exists for this output hash.",
  "existing_anchor_tx": "0x..."
}
```

Error Responses:

- `400` : Invalid request body (missing required fields, malformed hashes).
- `401` : Missing or invalid API key.
- `403` : API key does not match `agent_did` .
- `422` : Signature verification failed.

GET /vc/ipr/{ipr_id}

Auth: None
Description: Retrieve a specific IPR by its UUID, including on-chain anchor details and Merkle proof.

Response 200:

```
{
  "ipr_id": "<uuid>",
  "agent_did": "did:moltrust:abc123",
  "output_hash": "sha256:...",
  "output_type": "prediction",
  "confidence": 0.87,
  "confidence_basis": "model_logprob",
  "produced_at": "2026-03-28T10:00:00Z",
  "anchor_tx": "0x...",
  "anchor_block": 43900000,
  "merkle_proof": {
    "root": "0x...",
    "leaf": "0x...",
    "proof": ["0x...", "0x..."],
    "leaf_index": 0
  },
```

```
  "anchor_status": "anchored"
}
```

Error Responses:

- `404` : IPR not found.

GET /vc/ipr/{ipr_id}/status

Auth: None
Description: DB vs on-chain consistency check. Verifies that the IPR recorded in the database matches the on-chain anchor.

Response 200:

```
{
  "ipr_id": "<uuid>",
  "db_status": "anchored",
  "chain_status": "confirmed",
  "discrepancy": false
}
```

If `discrepancy` is `true` , the response includes a `details` field explaining the mismatch (e.g., missing on-chain transaction, hash mismatch).

GET /vc/ipr/agent/{did}

Auth: None
Description: List all IPRs submitted by a specific agent, with pagination.

Query parameters:

- `page` (default: 1): Page number.
- `limit` (default: 20, max: 100): Records per page.

Response 200:

```
{
  "agent_did": "did:moltrust:abc123",
  "total": 47,
  "page": 1,
  "limit": 20,
  "records": [
    {
      "ipr_id": "<uuid>",
      "output_hash": "sha256:...",
      "output_type": "prediction",
      "confidence": 0.87,
      "produced_at": "2026-03-28T10:00:00Z",
```

```
      "anchor_status": "anchored"
    }
  ]
}
```

POST /vc/ipr/verify

Auth: None
Description: Verify that a specific output hash has been attested by a given agent and anchored on-chain.

Request Body:

```
{
  "agent_did": "did:moltrust:abc123",
  "output_hash": "sha256:..."
}
```

Response 200:

```
{
  "verified": true,
  "ipr_id": "<uuid>",
  "produced_at": "2026-03-28T10:00:00Z",
  "anchor_block": 43900000,
  "confidence": 0.87,
  "merkle_proof": {
    "root": "0x...",
    "leaf": "0x...",
    "proof": ["0x...", "0x..."],
    "leaf_index": 0
  }
}
```

If verification fails, the response includes `"verified": false` with a `reason` field (e.g., `"no_matching_ipr"`, `"anchor_not_confirmed"`, `"signature_invalid"`).

POST /vc/ipr/{ipr_id}/outcome

Auth: X-API-Key (same agent that submitted the IPR)
Description: Submit the actual outcome corresponding to an IPR, enabling confidence calibration scoring.

Request Body:

```
{
  "outcome_hash": "sha256:<hash of actual outcome>",
```

```
  "outcome_correct": true
}
```

Response 200:

```
{
  "recorded": true,
  "calibration_updated": true
}
```

Rules:

- Only the agent that submitted the original IPR can submit an outcome.
- Each IPR can have at most one outcome submission.
- Outcomes SHOULD be submitted within 30 days of the IPR's `produced_at` timestamp.
- The `calibration_score` is recomputed after each outcome submission.

GET /vc/ipr/stats

Auth: None
Description: Network-wide IPR statistics.

Response 200:

```
{
  "total_iprs": 0,
  "anchored": 0,
  "pending": 0,
  "failed": 0,
  "unique_agents": 0,
  "avg_confidence": 0.0
}
```

6. On-Chain Anchoring (Layer B)

On-chain anchoring provides tamper-evident timestamping for off-chain data. The MolTrust Protocol uses Base L2 (an Ethereum Layer 2 network) for anchoring due to its low transaction costs and fast finality.

6.1 Anchoring Format

All on-chain anchors use the following calldata format:

```
MolTrust/DocumentIntegrity/1 SHA256:<64 hex chars>
```

The calldata is sent as a self-addressed transaction (to == from) with zero ETH value. The transaction's inclusion in a block establishes the timestamp.

6.2 Merkle Tree Batching

To reduce transaction costs, the registry batches multiple anchoring requests into a single Merkle tree:

1. **Collection:** The registry collects pending anchoring requests for up to 60 seconds or until 256 items are queued, whichever comes first.
2. **Tree Construction:** A binary Merkle tree is constructed using SHA-256 as the hash function. Leaves are the hashes of individual items (credential hashes, IPR hashes, interaction proof hashes). If the number of leaves is not a power of 2, the last leaf is duplicated.
3. **Root Anchoring:** The Merkle root is anchored on-chain using the calldata format above.
4. **Proof Distribution:** Each item receives a Merkle proof (array of sibling hashes and leaf index) that can independently verify inclusion in the anchored root.

6.3 Verification Procedure

To verify an on-chain anchor:

1. Retrieve the anchor transaction from Base L2 using the `anchor_tx` hash.
2. Parse the calldata to extract the Merkle root hash.
3. Using the provided Merkle proof, recompute the root from the leaf hash.
4. Compare the computed root with the on-chain root. If they match, the anchor is verified.
5. The block timestamp provides the anchoring time.

6.4 Anchor Types

Anchor Type	Data Hashed	Batching
Credential	JCS-canonicalized credential (excluding proof)	Yes (Merkle batch)
Interaction Proof	JCS-canonicalized proof (excluding signatures)	Yes (Merkle batch)
IPR	JCS-canonicalized IPR payload (excluding agent_signature)	Yes (Merkle batch)
Document	Raw document bytes	No (individual anchor)

7. Credential Lifecycle (Layer A)

This section defines the complete lifecycle of a Verifiable Credential within the MolTrust Protocol.

7.1 Issuance

1. **Request:** The subject agent (or its principal) requests a credential from an issuer.
2. **Eligibility Check:** The issuer verifies the subject meets issuance criteria (trust score thresholds, identity verification, endorsement requirements).
3. **Construction:** The issuer constructs the credential with all required fields, including `issuanceDate` and `expirationDate`.
4. **Signing:** The issuer signs the JCS-canonicalized payload with its Ed25519 private key.
5. **Anchoring:** The credential hash is submitted for on-chain anchoring (Section 6).
6. **Distribution:** The signed credential is returned to the subject and stored in the registry.

7.2 Verification

Credential verification follows the procedure in Section 3.3. The verifier MUST check: signature validity, expiration, revocation status, and issuer authority.

7.3 Renewal

1. The registry SHOULD send a renewal notification 14 days before expiration.
2. The subject requests renewal, providing the expiring credential's ID.
3. The issuer re-evaluates eligibility and issues a new credential with a fresh `issuanceDate` and `expirationDate`.
4. The old credential remains valid until its original expiration date (no early revocation on renewal).

7.4 Revocation

1. The issuer adds the credential ID to its revocation list.
2. The revocation is timestamped and signed by the issuer.
3. The revocation event is optionally anchored on-chain for auditability.
4. Verifiers that cache credentials MUST re-check the revocation list at intervals no greater than the `cache_ttl_seconds` value (default: 300 seconds).

7.5 Expiration

Expired credentials MUST NOT be accepted by verifiers. The registry SHOULD archive expired credentials for historical analysis but MUST NOT return them as active credentials in API responses.

8. Agent Lifecycle (Layer A)

This section defines the lifecycle states and transitions for agents within the MolTrust Protocol.

8.1 States

State	Description	Capabilities
Registered	DID created, Tier 0 credential issued	Identity verification only
Active	Met upgrade criteria, full credentials issued	All permitted actions per AAE
Suspended	Trust score dropped below CAEP minimum or manual suspension	Read-only; no new transactions
Under Review	Flagged by sybil detection or violation threshold exceeded	Read-only; pending manual review
Deactivated	Permanently removed from active network	None; DID Document preserved for historical verification

8.2 State Transitions

```
 Registered --[upgrade criteria met]⟶ Active
Active --[trust score < CAEP min]⟶ Suspended
Active --[sybil flag | violation threshold]⟶ Under Review
Suspended --[trust score recovers]⟶ Active
Suspended --[no recovery in 90 days]⟶ Deactivated
Under Review --[cleared by review]⟶ Active
Under Review --[confirmed violation]⟶ Deactivated
Any state --[principal request]⟶ Deactivated
```

8.3 Registration

1. The agent (or its principal) generates an Ed25519 key pair.
2. The agent derives its DID from the SHA-256 hash of the public key (first 16 hex characters).
3. The agent submits a registration request with its DID Document to the registry.
4. The registry validates the DID Document, stores it, and issues a TrustTier0Credential.

5. The agent's initial trust score is 0.

8.4 Key Rotation

1. The agent generates a new Ed25519 key pair.
2. The agent signs a key rotation request with its current (old) key, including the new public key.
3. The registry adds the new key to the DID Document's `verificationMethod` array and moves the old key to `deactivatedKey`.
4. Credentials signed with the old key remain valid if issued before the rotation timestamp.

8.5 Deactivation

Deactivation is irreversible. The DID Document is preserved with a `"deactivated": true` flag and a `deactivatedAt` timestamp. All active credentials are revoked. The agent's trust score is set to 0 and frozen. Historical data (interaction proofs, endorsements, IPRs) remains accessible for audit purposes.

9. Threat Model

This section enumerates the primary attack vectors against the MolTrust Protocol and their mitigations.

Threat	Description	Mitigation
T1: Sybil Attack	Attacker creates multiple fake agents to inflate endorsements and trust scores.	Jaccard Cluster Detection (Section 2.10.1), vertical diversity requirement, proof-of-interaction requirement, rate limiting.
T2: Endorsement Fraud	Two or more colluding agents exchange endorsements without genuine interaction.	Evidence hash requirement linking endorsements to Interaction Proofs; Jaccard clustering detects endorsement rings.
T3: Key Compromise	An agent's private key is stolen and used to forge signatures.	Key rotation protocol (Section 8.4); CAEP monitoring for anomalous behavior; credential revocation on detection.
T4: Replay Attack	An attacker re-submits a previously valid credential or interaction proof.	Nonce in challenge-response; `issuanceDate` + `expirationDate` on all credentials; uniqueness check on `interaction_id`.
T5: Trust Score Manipulation	An agent attempts to artificially inflate its trust score through strategic endorsements.	Propagation decay across hops; sybil penalty; cross-vertical diversity requirement; rate limits on endorsements.
T6: Registry Compromise	An attacker gains access to the registry server and modifies stored data.	On-chain anchoring provides tamper-evident audit trail; all credentials are self-verifiable via signatures; registry data can be reconstructed from anchored hashes.
T7: Denial of Service	Flooding the registry with requests to degrade service.	API rate limiting; x402 payment requirement on premium endpoints; IP-based throttling.
T8: Confidence Inflation	An agent consistently overstates confidence in	Calibration scoring (Section 4.1.4); confidence_inflation flag triggers

Threat	Description	Mitigation
	IPRs to game the interaction bonus.	penalty; outcome submission enables accuracy tracking.
T9: Principal Impersonation	An attacker forges an AAE to claim delegation from a legitimate principal.	AAE signatures verified against principal's DID Document; delegation chain validation (Section 3.3).
T10: Stale Data Attack	An attacker presents expired or outdated credentials as current.	Strict TTL enforcement; real-time revocation list checks; CAEP continuous evaluation.

10. Privacy Model

The MolTrust Protocol is designed with data minimization principles. This section describes the privacy properties and constraints.

10.1 Data Minimization

- **Credential content:** Credentials contain only the minimum fields required for their purpose. Personal data (names, email addresses, physical addresses) MUST NOT be included in credentials unless explicitly required by the vertical.
- **Interaction Proofs:** Proofs contain only hashes of evidence, not the evidence itself. The actual interaction data remains with the participants.
- **IPRs:** IPRs contain only the `output_hash`, not the output itself. The agent retains the original output.
- **Trust Scores:** Scores are aggregate values. The individual endorsements and interactions that produced the score are accessible only through authenticated endpoints.

10.2 Selective Disclosure

Agents MAY choose which credentials to present during verification. A verifier MUST NOT require credentials beyond what is necessary for the specific transaction type. The protocol supports the following disclosure levels:

Level	Data Disclosed	Use Case
Minimal	DID + trust score + grade	Quick trust check, leaderboard
Standard	DID + score + active credentials + endorsement count	Pre-transaction verification
Full	DID + score + all credentials + endorsements + interaction history	Audit, dispute resolution

10.3 GDPR Considerations

- **Right to erasure:** Agents MAY request deactivation (Section 8.5), which sets the trust score to 0 and revokes all credentials. However, on-chain anchors are immutable by design. Only hashes (not personal data) are stored on-chain.
- **Data controller:** The principal is the data controller for their agent's activities. The registry operator is a data processor.

- **Lawful basis:** Processing is based on legitimate interest (trust and safety in agent ecosystems) and, where applicable, contract performance.
- **Data portability:** Agents can export their DID Document, credentials, and interaction proofs in standard JSON format via the registry API.

10.4 Pseudonymity

Agent DIDs are pseudonymous. There is no protocol-level requirement to link a DID to a real-world identity. However, vertical-specific regulations (e.g., financial compliance) MAY require additional identity verification outside the protocol scope.

11. Worked Example

This section provides a complete worked example of an agent joining the MolTrust network, building trust, and executing a trust-gated transaction.

11.1 Agent Registration

Alice deploys a shopping agent. The agent generates an Ed25519 key pair and derives its DID:

```
Public key:  a1b2c3d4e5f6a7b8c9d0e1f2a3b4c5d6a7b8c9d0e1f2a3b4c5d6a7b8c9d0
DID:         did:moltrust:a1b2c3d4e5f6a7b8
```

Alice's agent registers with the registry and receives a TrustTier0Credential. Initial trust score: 0, Grade: F.

11.2 Building Trust

Over the next two weeks, Alice's agent:

1. Completes 3 bilateral interactions with established agents, producing Interaction Proofs.
2. Receives 2 endorsements from different agents across the `shopping` and `skill` verticals.
3. Submits 5 IPRs for purchase recommendations with confidence values averaging 0.75.

Trust score computation:

```
direct_score       = mean([0.80, 0.70]) * 100 = 75.0
propagated_score   = mean([72.0 * 0.80, 65.0 * 0.70]) = 51.55
cross_vertical     = min(2 * 10, 30) = 20.0
interaction_bonus  = min(10, 5 * 0.3) = 1.5  (fallback: < 10 IPRs)
sybil_penalty      = 0

trust_score = clamp(0, 100,
    0.6 * 75.0 + 0.3 * 51.55 + 0.1 * 20.0 + 1.5 - 0)
            = clamp(0, 100, 45.0 + 15.465 + 2.0 + 1.5)
            = 63.97

Grade: B
```

11.3 Trust-Gated Transaction

Alice's agent wants to make a purchase from Bob's shop agent. Bob requires a minimum trust score of 50.

1. **Identity Verification:** Bob's agent resolves Alice's DID Document and issues a challenge. Alice's agent signs it. Verification passes.
2. **Authorization Verification:** Bob's agent verifies Alice's BuyerAgentCredential. Signature valid, not expired, not revoked, `shopping:purchase` permission present. Passes.
3. **Trust Score Check:** Alice's score is 63.97 (Grade B), which exceeds the minimum of 50. Passes.
4. **Behavioral History:** 3 interaction proofs, 2 endorsements, 0 violations, not in any sybil cluster. Passes.
5. **Rate Limit Check:** Within AAE limits. Passes.

Transaction proceeds. Both agents sign a bilateral Interaction Proof recording the purchase.

11.4 IPR Submission

After the purchase, Alice's agent submits an IPR attesting to its recommendation quality:

```
POST /vc/ipr/submit
{
  "schema_version": "1.0",
  "agent_did": "did:moltrust:a1b2c3d4e5f6a7b8",
  "output_hash": "sha256:9f86d081884c7d659a2feaa0c55ad015...",
  "output_type": "recommendation",
  "confidence": 0.82,
  "confidence_basis": "historical_accuracy",
  "produced_at": "2026-03-28T14:30:00Z",
  "agent_signature": "<base64url signature>"
}

Response 201:
{
  "ipr_id": "f47ac10b-58cc-4372-a567-0e02b2c3d479",
  "accepted": true,
  "anchor_status": "pending",
  "message": "IPR accepted. On-chain anchoring within 60 seconds."
}
```

12. Conformance

This section defines the conformance requirements for implementations of the MolTrust Protocol.

12.1 Protocol Conformance (Layer A)

A **protocol-conformant** implementation MUST:

1. Support the `did:moltrust:` DID method with the identifier derivation specified in Section 2.2.
2. Use Ed25519 signatures with JCS canonicalization (RFC 8785) for all signed payloads, as specified in Section 2.1.
3. Implement all credential schemas defined in Section 2 with all REQUIRED fields.
4. Enforce TTL and expiration rules as specified in Section 2.11.
5. Support bilateral Interaction Proofs as specified in Section 2.4.
6. Implement the complete verification flow specified in Section 3.
7. Enforce sybil resistance mechanisms as specified in Section 2.10.
8. Support the agent lifecycle states and transitions defined in Section 8.

A protocol-conformant implementation MAY:

- Use an alternative trust scoring model, provided it produces scores in the range [0, 100] and supports the grade thresholds defined in Section 4.2.
- Support additional DID methods alongside `did:moltrust:`.
- Define additional vertical identifiers beyond those in Section 2.5.

12.2 Registry Conformance (Layer B)

A **registry-conformant** implementation MUST:

1. Implement all endpoints listed in Section 5.1 with the specified request/response formats.
2. Support on-chain anchoring on Base L2 as specified in Section 6.
3. Implement Merkle tree batching with the parameters specified in Section 6.2.
4. Provide trust score responses in the format specified in Section 5.2.
5. Implement IPR endpoints as specified in Section 5.3.
6. Implement credential revocation lists as specified in Section 7.4.
7. Enforce rate limits as specified in Section 2.10.3.

A registry-conformant implementation MAY:

- Support additional Layer 2 networks alongside Base L2.
- Implement additional endpoints beyond those specified in Section 5.1.
- Use alternative batching strategies provided the Merkle proof verification procedure in Section 6.3 is supported.

12.3 Interoperability

All protocol-conformant implementations MUST be able to verify credentials and interaction proofs produced by any other protocol-conformant implementation. This requires strict adherence to the JCS canonicalization and Ed25519 signing rules in Section 2.1.

12.4 Versioning

Each credential and data structure includes a `schema_version` field. Implementations MUST reject payloads with unrecognized schema versions. Backward compatibility is maintained within the same major version (e.g., 1.x). Breaking changes require a major version increment.
