PoCI Specification v0

Proof of Correct Inference – RFC■Style Standard Document

1. INTRODUCTION

PoCI (Proof of Correct Inference) is an open, neutral and immutable standard for verifying the integrity of AI inferences using cryptographic commitments, canonical hashing, deterministic serialization and digital signatures. The goal of PoCI is to provide a transparent, auditable and secure foundation for AI verification across high■risk and regulated environments, without exposing input data, output data or proprietary model internals.

PoCI v0 defines the minimal normative core of the standard. It establishes the fundamental data structures, the required cryptographic primitives, the invariants that MUST be respected, and the flows for producing, validating and auditing inference commits. Future versions of the standard extend this foundation with dispute resolution, verifiable compute integration (BOS/BitSNARK) and optional ZKML proofs.

2. TERMINOLOGY

Model — A computational system that produces an output given an input.

Inference — The execution of the model on a provided input.

Provider — Entity performing the inference and generating the signed PoCI commit.

Verifier — Entity validating a PoCI commit according to the specification.

Commitment — Cryptographic hash representing data without revealing the underlying content.

PoCIModelDescriptor — Structure describing a model using commitments rather than model internals.

PoCIInferenceCommit — The signed, immutable record representing an inference.

inputHash — SHA■256 hash of the canonical representation of the input.

outputHash — SHA■256 hash of the canonical representation of the output.

commitId — SHA■256 hash of the canonical form of the commit without the signature.

Signature — Digital signature created by the Provider over commitId.

Challenge — Object representing a dispute of a commit (non■functional in v0).

Version — Version string indicating the PoCI specification version.

3. NORMATIVE DATA STRUCTURES

This section defines the mandatory structures that constitute the PoCI core. Implementations MUST follow these structures and MUST reject any object that violates the required fields or constraints.

3.1 PoCIModelDescriptor

PoCIModelDescriptor MUST contain the following fields: - modelId: string - modelVersion: string - weightsCommitment: hex■string - architectureCommitment: hex■string - forwardCodeCommitment: hex■string - providerPublicKey: string (PEM or hex■encoded) All fields are REQUIRED. Implementations MUST reject descriptors that contain unknown mandatory fields, omit any required field or fail canonical serialization. Descriptors are immutable once published.

3.2 PoCIInputDescriptor

PoCIInputDescriptor MUST contain: - inputHash: hex■string Optional: - context: enum(BANKING | HEALTH | LEGAL | OTHER) - meta: object inputHash MUST be generated by the client or trusted external component.

3.3 PoCIInferenceCommit

A valid PoCIInferenceCommit MUST contain: - commitId: hex■string - modelDescriptorRef: string - inputHash: hex■string - outputHash: hex■string - timestamp: ISO■8601 - proverPublicKey: string - proverSignature: base64■string - version: "poci-v0" Any modification of fields after signature creation invalidates the commit.

3.4 PoCIChallenge (v0 conceptual)

A challenge MUST include: - challengeId - commitId - reason (enum) - challengerPublicKey - createdAt PoCI v0 does NOT define dispute execution. BOS integration appears in PoCI v1.

4. SERIALIZATION AND HASHING RULES

All PoCI objects MUST be serialized using canonical JSON. Canonical JSON is defined as: - keys sorted lexicographically - no undefined values - UTF■8 encoding - minimal representation

commitId MUST be computed as: commitId = SHA■256(canonical_json(commit_without_signature))

inputHash and outputHash MUST be computed as: hash = SHA■256(canonical_json(data))

Signatures MUST be produced over commitId using RSA■2048 or ECDSA secp256k1. The verifier MUST confirm the signature and MUST reject commits where commitId does not match the canonical form.

5. INVARIANTS

Implementations MUST enforce the following normative invariants:

- inputHash MUST correspond to the canonical data.
- outputHash MUST correspond to the canonical output.
- commitId MUST be computed as specified.
- All commits MUST be signed.
- Signatures MUST validate against the provided public key.
- modelDescriptorRef MUST reference a valid descriptor.
- version MUST equal "poci■v0".
- REQUIRED fields MUST NOT be empty.
- Commits MUST be immutable once signed.

6. PROCESS FLOWS

6.1 Model Registration

Provider constructs commitments for model weights, architecture and forward code. Provider publishes PoCIModelDescriptor. Descriptors are immutable.

6.2 Inference and Commit Generation

Client generates inputHash. Provider executes model, computes outputHash, constructs commit body, computes commitId, and signs it. The resulting PoCIInferenceCommit is immutable.

6.3 Commit Verification

Verifier recomputes commitId, validates signature, checks invariants, validates timestamp and version. A commit is valid if and only if all normative rules pass.

6.4 Challenge (v0)

A challenge indicates concern about a commit. No dispute logic exists in v0. Future versions integrate BOS/BitSNARK for verifiable interactive dispute resolution.

7. VERSIONING RULES

PoCI uses incremental stable versioning. Breaking changes to serialization or commitId computation MUST generate a new major version. Implementations MUST remain backward■compatible with older versions.

8. FUTURE EXTENSIONS

PoCI v1 introduces BOS■based dispute resolution through BitSNARK dissection and Bitcoin■anchored arbitration. PoCI v2 introduces optional ZKML proof attachments for partial or full model verification. PoCI v3 introduces hybrid BOS+ZKML verification patterns.