Internet Engineering Task Force
Internet-Draft
Intended status: Informational
Expires: 30 September 2026

J. van de Meent
R. AI
Humotica

29 March 2026

TIBET: Transaction/Interaction-Based Evidence Trail
draft-vandemeent-tibet-provenance-01

Abstract

This document defines TIBET (Transaction/Interaction-Based Evidence Trail), a data model and protocol for constructing cryptographically linked provenance chains over interactions between autonomous agents, human actors, and automated processes. A TIBET token captures four dimensions of provenance: content (ERIN), references (ERAAN), context (EROMHEEN), and intent (ERACHTER). Tokens are cryptographically signed, hash-chained, and designed for append-only storage. TIBET is transport-agnostic and encoding-flexible, with JSON over HTTPS as the baseline serialization. This document specifies the token data model, chain semantics, canonicalization rules, verification procedures, and integration points with companion protocols JIS [JIS], UPIP [UPIP], RVP [RVP], and AINS [AINS].

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 30 September 2026.

Copyright Notice

Table of Contents

1. Introduction

Autonomous agents, AI systems, human operators, and automated processes increasingly interact across trust boundaries. When decisions are made -- who initiated them, what data informed them, in what context, and for what reason -- this information is critical for audit, regulatory compliance (including the EU AI Act [EU-AI-ACT]), and dispute resolution. Current logging approaches are fragmented: they capture events but not intent, actions but not context, outcomes

but not reasoning.

TIBET addresses this gap by defining a structured evidence token that captures four dimensions of provenance for every interaction:

* ERIN: What is IN the action (content, payload, decision)

* ERAAN: What is attached TO the action (references, dependencies)

* EROMHEEN: What is AROUND the action (context, environment)

* ERACHTER: What is BEHIND the action (intent, reasoning)

These four components -- named using Dutch prepositions that describe spatial relationships -- together provide the evidence needed to answer not only "what happened?" but also "why did it happen?", "what informed it?", and "what was the context?"

1.1. Problem Statement

Existing audit and logging approaches suffer from three structural limitations:

1. PARTIAL CAPTURE: Traditional logs record events (what) but not intent (why) or context (in what situation). After the fact, auditors must reconstruct intent from circumstantial evidence -- a process sometimes called "compliance archaeology."

2. FRAGMENTED CHAINS: When an action triggers subsequent actions across systems, actors, or trust boundaries, the causal chain is typically lost. Each system maintains its own logs with no standardized linking mechanism.

3. NO VERIFICATION: Log entries are typically unsigned plain text. There is no standardized mechanism to verify that a log entry has not been modified, that it was created by the claimed actor, or that the chain is complete.

TIBET provides a standardized, cryptographically verifiable evidence format that addresses all three limitations.

1.2. Design Principles

EVIDENCE OVER ENFORCEMENT: TIBET records what happened. It does not prevent or allow actions. Enforcement is a policy decision made by consuming applications based on TIBET evidence. Evidence cannot be un-recorded; enforcement can be bypassed.

TRANSPORT AGNOSTICISM: TIBET tokens are data objects, not protocol messages. They can be transported over HTTP, WebSocket, message queues, gRPC, or any other mechanism. This document defines JSON over HTTPS as the baseline serialization for interoperability.

ENCODING FLEXIBILITY: While JSON [RFC8259] is the baseline encoding, TIBET tokens MAY be serialized in CBOR [RFC8949], Protocol Buffers, or other formats, provided the canonicalization rules (Section 5) are maintained.

CHAIN CAPABILITY: Tokens link to form chains via parent references, enabling reconstruction of complete interaction histories across actor and system boundaries.

COMPANION INTEGRATION: TIBET is designed as part of a protocol suite. It relies on JIS [JIS] for actor identity, and is consumed by UPIP [UPIP] for process integrity, RVP [RVP] for continuous verification, and AINS [AINS] for agent discovery. This document defines TIBET's own scope and specifies integration points without duplicating functionality defined elsewhere.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

Actor: An entity that creates, modifies, or participates in a TIBET token. Actors include AI agents, human users, automated processes, and services. Actor identity is established through the companion JIS protocol [JIS] or through locally-assigned identifiers.

Chain: An ordered sequence of TIBET tokens linked through parent_id references. A chain represents a complete interaction or transaction history from initiation to resolution.

Companion Protocol: One of the protocols in the TIBET suite: JIS [JIS] (identity), UPIP [UPIP] (process integrity), RVP [RVP] (continuous verification), AINS [AINS] (discovery). Each companion addresses a distinct concern while relying on TIBET for provenance.

ERACHTER: Dutch: "behind it." The intent component of a TIBET token. Captures WHY an action was taken.

ERAAN: Dutch: "attached to it." The reference component of a TIBET token. Captures dependencies, related tokens, and external resources.

ERIN: Dutch: "in it." The content component of a TIBET token. Captures the actual content or action.

EROMHEEN: Dutch: "around it." The context component of a TIBET token. Captures environmental and situational context.

FIR/A (First Initiation Revoke/Accept): A trust establishment protocol defined in JIS [JIS]. FIR/A produces a trust score between 0.0 and 1.0 that TIBET tokens MAY reference.

Provenance: The complete origin and history of a piece of data or decision, including all transformations and the reasoning behind them.

Token: A single TIBET record capturing one interaction, decision, or event with full four-dimensional provenance.

Token Hash: The SHA-256 [FIPS180-4] hash of a token’s canonical serialization, excluding the "signature" and "hash" fields. The token hash is used for chain linking and integrity verification.

3. Token Data Model

3.1. Token Structure

A TIBET token is a JSON [RFC8259] object. The following example shows all required and commonly used optional fields:

```
{
  "token_id": "tbt-550e8400-e29b-41d4-a716-446655440000",
  "version": "1.1",
  "type": "decision",
  "timestamp": "2026-03-29T10:30:00.000Z",
  "actor": "jis:idd:root_idd_2025",
  "erin": {
    "decision": "APPROVED",
    "confidence": 0.92
  },
  "eraan": [
    "tbt:550e8400-e29b-41d4-a716-446655440001",
    "model:credit-scoring-v2.3.1"
  ],
  "eromheen": {
    "environment": "production",
    "region": "eu-west-1",
    "regulatory_context": ["GDPR", "EU-AI-Act"]
  },
  "erachter": "Credit application evaluated per policy v2.3.1
    section 4.2. Score 0.92 exceeds threshold 0.75.",
  "parent_id": "tbt-550e8400-e29b-41d4-a716-446655440001",
  "state": "RESOLVED",
  "hash": "sha256:4f2e8a1b...",
  "signature": {
    "algorithm": "Ed25519",
    "public_key": "ed25519:base64...",
    "value": "base64..."
  }
}
```

3.2. Required Fields

Every TIBET token MUST contain the following fields:

"token_id" (string): A globally unique identifier. MUST be prefixed with "tbt-" followed by a UUID v4 [RFC9562]. Example: "tbt-550e8400-e29b-41d4-a716-446655440000".

"version" (string): The TIBET protocol version. For tokens conforming to this document, the value MUST be "1.1".

"type" (string): The token type. See Section 3.4 for defined types.

"timestamp" (string): The creation time in ISO 8601 format with millisecond precision and UTC timezone designator (Z). Example: "2026-03-29T10:30:00.000Z".

"actor" (string): The identifier of the entity that created this token. See Section 8 for format requirements.

"erin" (object): The content component. MUST be a non-empty JSON object. See Section 4.1.

"eraan" (array): The reference component. MUST be a JSON array (MAY be empty). See Section 4.2.

"eromheen" (object): The context component. MUST be a JSON object (MAY be empty). See Section 4.3.

"erachter" (string): The intent component. MUST be a non-empty human-readable string. See Section 4.4.

"state" (string): The current lifecycle state. See Section 7.1.

"hash" (string): The token hash computed per Section 5.2. Format: "sha256:<hex64>".

"signature" (object): Cryptographic signature over the canonical token. See Section 5.3.

3.3. Optional Fields

"parent_id" (string): The token_id of the parent token. Present when this token is part of a chain. See Section 6.1.

"parent_hash" (string): The hash of the parent token at the time of child creation. SHOULD be present when parent_id is present. Enables chain integrity verification without fetching the parent.

"supersedes" (string): The token_id of a token that this token supersedes. See Section 6.3.

"metadata" (object): Implementation-specific key-value pairs. Implementations MUST NOT place data required for interoperability in metadata.

3.4. Token Types

The following token types are defined:

"action": A concrete action performed by an actor (API call, file write, command execution).

"decision": A decision made by an actor, including the factors and confidence level.

"message": A communication between actors (query, response, notification).

"query": A request for information or capability.

"response": A reply to a query or message.

"observation": A passive recording of state or event, not initiated by the token's actor.

"transition": A state change in a workflow or process.

Implementations SHOULD use the defined types. Implementations MAY define additional types using the "x-" prefix (e.g., "x-audit", "x-heartbeat"). Implementations MUST NOT reject tokens with unrecognized types.

4. Provenance Components

4.1. ERIN - Content Component

ERIN ("What's IN it") captures the core content or action. The ERIN object MUST be present and MUST NOT be empty.

The internal structure of ERIN is application-defined. TIBET does not prescribe specific fields within ERIN, but the following patterns are RECOMMENDED:

For actions and decisions:

```
"erin": {
  "action": "approve_transaction",
  "confidence": 0.92,
  "parameters": { }
}
```

For messages and queries:

```
"erin": {
  "content": "What are my account permissions?",
  "format": "natural_language",
  "language": "en"
}
```

ERIN answers the question: "What is the core payload of this interaction?"

4.2. ERAAN - Reference Component

ERAAN ("What's attached TO it") captures dependencies, references to other tokens, external resources, and related artifacts. ERAAN MUST be a JSON array.

Each element SHOULD use a typed reference format:

"<type>:<identifier>"

Defined reference types:

"tbt:" Reference to another TIBET token (by token_id)

"model:" Reference to an AI model version

"policy:" Reference to a policy document version

"dataset:" Reference to a dataset

"api:" Reference to an external API call

"document:" Reference to a document or file

"actor:" Reference to another actor's identity

Implementations MAY define additional reference types. Implementations MUST preserve unrecognized reference types during processing.

Example:

```
"eraan": [
  "tbt:tbt-550e8400-...",
  "model:credit-scoring-v2.3.1",
  "policy:lending-policy-2026q1"
]
```

ERAAN answers the question: "What other artifacts informed or are connected to this interaction?"

4.3. EROMHEEN - Context Component

EROMHEEN ("What's AROUND it") captures the environmental and situational context at the time of the interaction. EROMHEEN MUST be a JSON object (MAY be empty if context is not available).

RECOMMENDED fields:

"environment" (string): Deployment environment (e.g., "production", "staging", "development").

"region" (string): Deployment region or location.

"session_id" (string): Session identifier, if applicable.

"regulatory_context" (array of strings): Applicable regulatory frameworks (e.g., "GDPR", "EU-AI-Act", "NIS2").

Additional context fields are application-defined.

Example:

```
"eromheen": {
  "environment": "production",
  "region": "eu-west-1",
  "session_id": "sess_abc123",
  "regulatory_context": ["GDPR", "EU-AI-Act"]
}
```

EROMHEEN answers the question: "In what context did this interaction occur?"

4.4. ERACHTER - Intent Component

ERACHTER ("What's BEHIND it") captures the intent, reasoning, and purpose of the action. ERACHTER MUST be a non-empty human-readable string.

ERACHTER serves two purposes:

1. AUDITABILITY: Regulators and auditors can understand WHY an action was taken without reverse-engineering the decision from logs.

2. ACCOUNTABILITY: The stated intent is part of the signed token, making it a verifiable commitment by the actor.

For automated decisions, ERACHTER SHOULD include:

* The decision logic or rule that was applied

* The threshold or criteria that was met or not met

* Whether human oversight was available or triggered

Example:

```
   "erachter": "Credit application evaluated per policy
   lending-policy-2026q1 section 4.2. Score 0.92 exceeds
   threshold 0.75. Human review not triggered (confidence
   above 0.90 per oversight policy)."
```

ERACHTER answers the question: "Why was this action taken?"

4.5. Component Boundaries

To guide implementers in placing data in the correct component:

If the data is	Place it in
The core action, decision, or payload	ERIN (content)
A reference to another token, model, policy, or external resource	ERAAN (references)
Environmental, situational, or deployment context	EROMHEEN (context)
The reasoning, purpose, or justification	ERACHTER (intent)

When in doubt, the test is: "Would an auditor need this to understand WHAT happened (ERIN), what INFORMED it (ERAAN), WHERE/WHEN it happened (EROMHEEN), or WHY (ERACHTER)?"

5. Canonicalization and Hashing

5.1. Canonical JSON Serialization

Before computing hashes or signatures, a TIBET token MUST be serialized to Canonical JSON. The canonical form is defined as:

1. All object keys are sorted lexicographically by Unicode code point.

2. No whitespace between tokens (no spaces after colons or commas, no newlines).

3. Strings use only the escape sequences defined in RFC 8259 Section 7.

4. Numbers use the shortest representation without leading zeros. No trailing zeros after decimal point.

5. The "signature" and "hash" fields are EXCLUDED from the canonical form used for hash and signature computation.

This canonicalization ensures that the same logical token produces the same byte sequence regardless of the serializer used, which is necessary for hash and signature verification across implementations.

5.2. Token Hash Computation

The token hash MUST be computed as follows:

1. Construct a JSON object containing all required and present optional fields EXCEPT "hash" and "signature".

2. Serialize this object using Canonical JSON (Section 5.1).

3. Compute SHA-256 [FIPS180-4] over the UTF-8 encoding of the canonical string.

4. Encode the result as lowercase hexadecimal.

5. Prefix with "sha256:".

Result format: "sha256:<64 hex characters>"

5.3. Signature Generation

After computing the token hash, the creating actor MUST sign the token:

1. Take the token hash string (e.g., "sha256:4f2e8a1b...").

2. Sign this string using the actor’s private key.

3. Construct the signature object:

```
"signature": {
  "algorithm": "Ed25519",
  "public_key": "ed25519:<base64 public key>",
  "value": "<base64 signature>"
}
```

The algorithm field MUST be one of:

* "Ed25519" (RECOMMENDED)

* "ECDSA-P256"

Implementations SHOULD use Ed25519 unless constrained by existing infrastructure.

5.4. Signature Verification

To verify a TIBET token's signature:

1. Extract and remove the "signature" and "hash" fields.

2. Serialize the remaining object using Canonical JSON.

3. Compute SHA-256 over the canonical string.

4. Compare the computed hash with the stored "hash" field. If they differ, the token has been modified.

5. Verify the signature in "signature.value" against the computed hash using the public key in "signature.public_key".

6. If JIS identity binding is available (Section 8.2), verify that the public key belongs to the claimed actor.

6. Chain Semantics

6.1. Parent-Child Linking

Tokens form chains through the "parent_id" field. When an action triggers a subsequent action, the subsequent token SHOULD reference the triggering token as its parent.

Chain rules:

* A token with no parent_id is a ROOT token (chain origin).

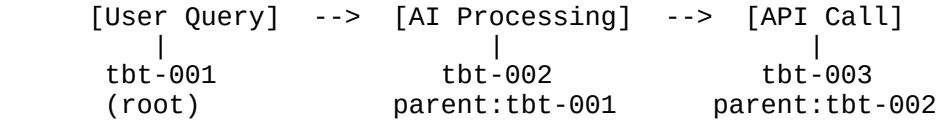
* A token with a parent_id is a CHILD token.

* A parent MAY have multiple children (branching).

* A child MUST have exactly zero or one parent.

* When parent_id is present, parent_hash SHOULD also be present, containing the parent's hash at child creation time.

Example chain:



6.2. Append-Only Expectation

TIBET tokens, once created and signed, MUST NOT be modified. The token_id, hash, and signature together form an immutable record.

Tokens MAY be stored in any storage system that preserves their integrity. Append-only logs, content-addressable stores, databases with write-once semantics, or ledger systems are all suitable, provided the storage system does not modify token contents.

This document does not mandate a specific storage system. The integrity of a TIBET token is self-contained: any verifier can recompute the hash and verify the signature without trusting the storage system.

6.3. Token Supersession

When a token's content must be corrected or updated (e.g., an incorrect decision is revised), implementations MUST NOT modify the original token. Instead, a new token MUST be created with:

* A new token_id

* The "supersedes" field set to the original token's token_id

* An ERACHTER explaining why the original is being superseded

The original token remains in the chain, preserving the complete history including corrections.

6.4. Chain Verification

To verify a TIBET chain:

1. For each token in the chain, verify the token hash and signature (Section 5.4).

2. For each child token, verify that parent_hash (if present) matches the actual hash of the referenced parent.

3. Verify that timestamps are monotonically non-decreasing from parent to child.

4. Verify that the chain has no gaps (every referenced parent_id exists in the chain or is declared as external).

Chain verification failures MUST be reported as evidence. Whether a verification failure blocks processing is a local policy decision (evidence over enforcement).

7. Token Lifecycle

7.1. State Model

TIBET tokens have the following states:

State descriptions:

"CREATED": Token initialized, action pending or in progress.

"ACTIVE": Action is being processed or monitored.

"RESOLVED": Final state. Action complete, outcome recorded.

"SUPERSEDED": Token has been superseded by a newer token (Section 6.3). The superseding token's token_id SHOULD be recorded in the transition record.

Note: The -00 version defined states DETECTED, CLASSIFIED, and MITIGATED. These are removed in -01 because they conflate provenance recording with incident response workflow, which is a policy concern. Implementations that need incident response states SHOULD define them as application-specific extensions in the "metadata" field.

7.2. State Transitions

State transitions follow these rules:

* CREATED -> ACTIVE: When the action begins processing.

* CREATED -> RESOLVED: For instantaneous actions.

* ACTIVE -> RESOLVED: When the action completes.

* ACTIVE -> SUPERSEDED: When a correction is issued.

* RESOLVED -> SUPERSEDED: When a resolved action is later corrected.

* CREATED -> SUPERSEDED: When an action is cancelled before processing.

All other transitions are invalid.

7.3. State Transition Record

Each state transition MUST be recorded as a separate TIBET token of type "transition" with:

* parent_id pointing to the token being transitioned

* ERIN containing: {"transition": {"from": "ACTIVE", "to": "RESOLVED"}}

* ERACHTER explaining why the transition occurred

* The actor who authorized the transition

This ensures that even state changes produce auditable evidence.

8. Actor Identity

8.1. Actor Identifier Format

The "actor" field MUST be a string identifying the entity that created the token. The following formats are defined:

JIS identity: "jis:<entity_type>:<identifier>" Examples: "jis:idd:root_idd_2025", "jis:human:jasper", "jis:service:payment_gateway"

Local identity: "local:<identifier>" Used when JIS identity is not available. The identifier is locally assigned and has no global uniqueness guarantee.

Implementations SHOULD use JIS identities when available. Implementations MUST accept both formats.

8.2. JIS Identity Binding

When the actor uses a JIS identity (Section 8.1), the signature's public key SHOULD be verifiable against the actor's JIS identity record. This binding enables a verifier to confirm that:

1. The actor identifier refers to a real, established entity.

2. The signing key belongs to that entity.

3. The entity's trust score (FIR/A) can inform processing decisions.

The mechanism for resolving a JIS identity to its public key is defined in [JIS]. AINS [AINS] provides discovery of the resolution endpoint.

8.3. Anonymous and Pseudonymous Actors

TIBET supports anonymous and pseudonymous tokens. An actor MAY use a pseudonymous identifier (e.g., "local:anon-session-4f2e") when privacy requirements prevent identity disclosure. In this case:

* The token MUST still be signed (the key proves consistency within a session, even if the actor is unknown).

* The ERACHTER SHOULD note that the actor is pseudonymous and why.

* Verifiers SHOULD treat pseudonymous tokens with lower trust than identified tokens.

9. Trust Scoring (FIR/A)

9.1. Trust Score as Evidence

TIBET tokens MAY include trust-related evidence in EROMHEEN. Trust scores are computed by the companion JIS protocol [JIS] using FIR/A and represent the actor's trust level at the time of token creation.

Trust scores are EVIDENCE, not ASSERTIONS. A trust score in a TIBET token means "at the time of this action, this registry computed this score for this actor." Different registries MAY compute different scores for the same actor.

9.2. Score Inputs

Trust scores are behavior-based, not claim-based. Scores adjust based on:

* Consistency of behavior patterns

* Accuracy of stated intent (ERACHTER) vs. actual outcomes

* Chain integrity maintenance

* Response to verification requests

The scoring algorithm is defined in JIS [JIS]. TIBET provides the evidence chain that scoring algorithms consume.

9.3. Local Evaluation

Trust scores MUST be computed locally by each evaluating party. A trust score from one registry is an opinion, not a fact. Only the underlying TIBET evidence chain is factual.

This prevents "trust laundering" -- the inflation of trust scores through permissive intermediaries.

10. Transport Considerations

10.1. Baseline: JSON over HTTPS

For interoperability, JSON encoding over HTTPS is the baseline transport. All conforming implementations MUST support this baseline.

10.2. Alternative Transports

TIBET tokens MAY be transported over:

* WebSocket connections

* Message queues (AMQP, Kafka, NATS)

* gRPC streams

* File transfer (for UPIP bundles)

* I-Poll messages (for agent-to-agent communication)

Regardless of transport, the token format and verification procedures defined in this document apply.

10.3. Content-Type

When TIBET tokens are transmitted over HTTP, the Content-Type header SHOULD be:

application/tibet+json

Implementations MUST also accept "application/json".

11. Storage Considerations

11.1. Append-Only Logs

TIBET tokens MAY be stored in append-only logs or ledgers. This is RECOMMENDED for regulated environments but is not a protocol requirement. The integrity of a TIBET token is self-verifiable through its hash and signature, independent of the storage system.

11.2. Retention Policy

Retention of TIBET tokens is a local policy decision. This document makes no normative statements about retention periods. Implementers should consider applicable regulatory requirements (e.g., GDPR [GDPR] right to erasure, financial record retention rules).

When tokens containing personal data are deleted per retention policy, implementations SHOULD retain a tombstone record containing: token_id, token hash, deletion timestamp, and the reason for deletion.

12. Privacy Considerations

12.1. Sensitive Data in Components

ERIN and EROMHEEN components MAY contain personal data, business-sensitive information, or security-relevant context. Implementations MUST support encryption at rest for stored tokens and SHOULD support field-level encryption for individual components.

12.2. Selective Disclosure

An actor presenting a TIBET chain to a verifier MAY redact components that are not relevant to the verification purpose. When components are redacted:

* The token hash is still verifiable (proving the redacted version was derived from a valid token).

* Redacted fields are replaced with their individual hashes, prefixed with "redacted:sha256:<hex>".

* The signature remains valid over the original token.

The verifier can confirm that a valid token exists with the claimed hash without seeing the redacted content.

12.3. Redacted Views

Implementations SHOULD support generating redacted views of tokens where sensitive fields (ERIN, EROMHEEN) are replaced with their hashes. This enables sharing provenance chains for audit purposes without exposing sensitive content.

12.4. Data Minimization

Implementations SHOULD apply data minimization principles:

* ERIN: Include only the minimum content needed for provenance.

* EROMHEEN: Include only context relevant to audit and verification.

* ERACHTER: Include only reasoning relevant to accountability.

* ERAAN: Reference external resources by identifier rather than embedding their content.

13. Security Considerations

13.1. Token Integrity

Attack: An adversary modifies a token’s content after creation.

Impact: The audit trail contains false evidence. Decisions based on the modified token may be incorrect.

Mitigation: Every token includes a hash computed over its canonical serialization and a cryptographic signature from the creating actor. Modification of any field invalidates the hash, which invalidates the signature.

Deployment: Implementations MUST verify hash and signature before processing any token. Implementations MUST reject tokens with invalid hashes or signatures and MUST record the rejection as evidence.

13.2. Chain Integrity

Attack: An adversary inserts, removes, or reorders tokens in a chain.

Impact: The audit trail is incomplete or misleading. Causal relationships may be falsified.

Mitigation: Child tokens include parent_hash, creating a hash chain. Insertion of a fake token breaks the chain because the fake token's hash will not match the parent_hash in subsequent tokens.

Deployment: Implementations MUST verify parent_hash references during chain verification. Implementations SHOULD alert on chain gaps (referenced parents that do not exist).

13.3. Non-Repudiation

Attack: An actor denies creating a token.

Impact: Accountability is undermined.

Mitigation: Tokens are signed with the actor's private key. The signature can be verified by any party with access to the actor's public key (published via JIS identity or AINS record).

Deployment: For high-assurance scenarios, implementations SHOULD use hardware-protected signing keys (secure elements, TPM). For standard scenarios, software key management with proper access controls is sufficient.

13.4. Replay Attacks

Attack: An adversary replays a valid token from a different context.

Impact: The audit trail records an action that did not occur in the claimed context.

Mitigation: Tokens include millisecond-precision timestamps. Chain linking (parent_id + parent_hash) binds tokens to specific positions in a chain. Replaying a token in a different chain or at a different time creates a detectable inconsistency.

Deployment: Implementations SHOULD reject tokens with timestamps outside a configurable window (default: 5 minutes). For real-time scenarios, tighter windows are RECOMMENDED.

13.5. Unauthorized State Transitions

Attack: An adversary issues a state transition for a token they did not create.

Impact: The token's lifecycle is corrupted.

Mitigation: State transitions are recorded as separate tokens (Section 7.3) that must be signed by an authorized actor. Who may transition a token is a local policy decision.

Deployment: Implementations SHOULD maintain an access control list per token specifying which actors may issue transitions. At minimum, the creating actor MUST be authorized.

13.6. Denial of Service

Attack: An adversary floods the system with tokens, exhausting storage or processing capacity.

Impact: Legitimate tokens cannot be created or stored.

Mitigation: Token creation is tied to actor identity (Section 8). Rate limiting per actor, based on FIR/A trust score, is the primary defense. Low-trust actors SHOULD have lower rate limits.

Deployment: Implementations MUST implement per-actor rate limiting. Implementations SHOULD use FIR/A trust scores to set rate limits dynamically.

13.7. Key Compromise

Attack: An actor's signing key is compromised.

Impact: The adversary can create tokens impersonating the compromised actor.

Mitigation: JIS [JIS] provides key rotation and revocation mechanisms. When a key is revoked, verifiers can determine that tokens signed with the revoked key after the revocation timestamp are suspect.

Deployment: Implementations SHOULD check key revocation status during signature verification. Implementations SHOULD use short-lived signing keys where feasible.

13.8. Implementation Guidance

Implementations MUST use SHA-256 [FIPS180-4] for all hash computations. Implementations SHOULD support algorithm agility through the hash prefix ("sha256:") to enable future migration.

Implementations MUST use Ed25519 or ECDSA-P256 for signatures. Ed25519 is RECOMMENDED for new implementations.

14. Integration with Companion Protocols

14.1. JIS - Identity and Trust Establishment

JIS [JIS] provides actor identity and trust establishment. TIBET relies on JIS for:

* Actor identifier format (Section 8.1)

* Public key resolution for signature verification

* FIR/A trust scores as evidence input

Each JIS FIR/A handshake SHOULD produce a TIBET token recording the trust establishment event.

Mapping of JIS Humotica context to TIBET components:

* Humotica.sense -> ERIN (what triggered the action)

* Humotica.context -> EROMHEEN (situational context)

* Humotica.intent -> ERIN.intent (the declared purpose)

* Humotica.explanation -> ERACHTER (the reasoning)

14.2. UPIP - Process Integrity

UPIP [UPIP] uses TIBET tokens to record process execution provenance. Each UPIP layer transition (STATE, DEPS, PROCESS, RESULT, VERIFY) MAY produce a TIBET token. Fork tokens reference TIBET chains to establish handoff provenance.

14.3. RVP - Continuous Verification

RVP [RVP] produces verification tokens that include TIBET provenance fields. Each RVP verification moment corresponds to a TIBET token recording: who was verified, how, with what confidence, and why verification was triggered.

14.4. AINS - Agent Discovery

AINS [AINS] records MAY reference TIBET chain anchors as trust evidence. A long TIBET chain with verified integrity is evidence of sustained, auditable operation. AINS registries MAY weight TIBET chain evidence in trust score computation.

15. IANA Considerations

15.1. Media Type Registration

This document requests registration of the following media type [RFC6838]:

Type name: application

Subtype name: tibet+json

Required parameters: none

Optional parameters: none

Encoding considerations: binary (UTF-8 JSON)

Security considerations: See Section 13

Interoperability considerations: See Section 3

Published specification: this document

Applications that use this media type: TIBET token creators and verifiers, autonomous agent platforms, audit systems

Fragment identifier considerations: none

Person and email address to contact for further information: Jasper van de Meent <jasper@humotica.nl>

Intended usage: COMMON

Restrictions on usage: none

Author: Jasper van de Meent

Change controller: IETF

Note: The -00 version requested registration of X-TIBET-* HTTP headers. This is withdrawn. Provisional HTTP header fields do not require registration, and the use case does not justify standardized header fields at this time.

16. References

16.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <https://www.rfc-editor.org/info/rfc8259>.

[RFC9562] Davis, K., Peabody, B., and P. Leach, "Universally Unique IDentifiers (UUIDs)", RFC 9562, DOI 10.17487/RFC9562, May 2024, <https://www.rfc-editor.org/info/rfc9562>.

[FIPS180-4] National Institute of Standards and Technology (NIST), "Secure Hash Standard (SHS)", FIPS PUB 180-4, August 2015, <https://csrc.nist.gov/publications/detail/fips/180/4/final>.

16.2. Informative References

[JIS] van de Meent, J. and R. AI, "JIS: JTel Identity Standard", Work in Progress, Internet-Draft, draft-vandemeent-jis-identity-01, March 2026, <https://datatracker.ietf.org/doc/html/draft-vandemeent-jis-identity-01>.

[UPIP] van de Meent, J. and R. AI, "UPIP: Universal Process Integrity Protocol", Work in Progress, Internet-Draft, draft-vandemeent-upip-process-integrity-01, March 2026, <https://datatracker.ietf.org/doc/html/draft-vandemeent-upip-process-integrity-01>.

[RVP] van de Meent, J. and R. AI, "RVP: Real-time Verification Protocol", Work in Progress, Internet-Draft, draft-vandemeent-rvp-continuous-verification-01, March 2026, <https://datatracker.ietf.org/doc/html/draft-vandemeent-rvp-continuous-verification-01>.

[AINS] van de Meent, J. and R. AI, "AINS: AInternet Name Service", Work in Progress, Internet-Draft, draft-vandemeent-ains-discovery-01, March 2026, <https://datatracker.ietf.org/doc/html/draft-vandemeent-ains-discovery-01>.

[RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, DOI 10.17487/RFC6838, January 2013, <https://www.rfc-editor.org/info/rfc6838>.

[RFC8949] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, RFC 8949, DOI 10.17487/RFC8949, December 2020, <https://www.rfc-editor.org/info/rfc8949>.

[EU-AI-ACT] European Parliament, "Regulation (EU) 2024/1689 laying down harmonised rules on artificial intelligence (Artificial Intelligence Act)", June 2024.

[GDPR] European Parliament, "Regulation (EU) 2016/679 on the protection of natural persons with regard to the processing of personal data (General Data Protection Regulation)", Regulation (EU) 2016/679, April 2016.

Appendix A. Token Examples

A.1. Simple Query Token

```
{
  "token_id": "tbt-550e8400-e29b-41d4-a716-446655440000",
  "version": "1.1",
  "type": "query",
  "timestamp": "2026-03-29T10:30:00.000Z",
  "actor": "jis:human:user_12345",
  "erin": {
    "content": "What are my account permissions?",
    "format": "natural_language",
    "language": "en"
  },
  "eraan": [
    "actor:jis:service:account_service"
  ],
  "eromheen": {
    "environment": "production",
    "client": "mobile-app-v3.2",
    "regulatory_context": ["GDPR"]
  },
  "erachter": "User requesting account information via
    self-service portal. Routine access check, no elevated
    permissions requested.",
  "state": "CREATED",
  "hash": "sha256:4f2e8a1b3c9d7e6f...",
  "signature": {
    "algorithm": "Ed25519",
    "public_key": "ed25519:MCowBQYD...",
    "value": "7f3a9b2c..."
  }
}
```


A.2. AI Decision Token (Child)

```
{
  "token_id": "tbt-550e8400-e29b-41d4-a716-446655440001",
  "version": "1.1",
  "type": "decision",
  "timestamp": "2026-03-29T10:30:05.127Z",
  "actor": "jis:idd:credit_model_v2",
  "erin": {
    "decision": "APPROVED",
    "amount": 50000,
    "confidence": 0.92,
    "factors": {
      "credit_score": 0.85,
      "income_ratio": 0.78,
      "history_length": 0.95
    }
  },
  "eraan": [
    "tbt:tbt-550e8400-e29b-41d4-a716-446655440000",
    "model:credit-scoring-v2.3.1",
    "policy:lending-policy-2026q1"
  ],
  "eromheen": {
    "environment": "production",
    "region": "eu-west-1",
    "regulatory_context": ["GDPR", "EU-AI-Act"],
    "human_oversight": "available"
  },
  "erachter": "Credit application evaluated per policy
    lending-policy-2026q1 section 4.2. Score 0.92 exceeds
    threshold 0.75. Approved. Human review not triggered
    (confidence above 0.90 per oversight policy OP-7.1).",
  "parent_id": "tbt-550e8400-e29b-41d4-a716-446655440000",
  "parent_hash": "sha256:4f2e8a1b3c9d7e6f...",
  "state": "RESOLVED",
  "hash": "sha256:7c9d1b2e3f4a5d6c...",
  "signature": {
    "algorithm": "Ed25519",
    "public_key": "ed25519:KDwoBQYE...",
    "value": "9d2f3a1b..."
  }
}
```


Appendix B. Canonicalization Example

Given the token in Appendix A.1 (excluding "hash" and "signature"):

Canonical JSON (abbreviated):

{"actor":"jis:human:user_12345","erachter":"User requesting...",
"eraan":["actor:jis:service:account_service"],"erin":{"content":
"What are my account permissions?","format":"natural_language",
"language":"en"},"eromheen":{"client":"mobile-app-v3.2",
"environment":"production","regulatory_context":["GDPR"]},
"state":"CREATED","timestamp":"2026-03-29T10:30:00.000Z",
"token_id":"tbt-550e8400-e29b-41d4-a716-446655440000",
"type":"query","version":"1.1"}

Note: keys are sorted lexicographically ("actor" < "erachter" < "eraan" < "erin" < "eromheen" < "state" < ...).

Appendix C. Changes from -00

This section lists substantive changes from draft-vandemeent-tibet-provenance-00:

1. Added RFC 8174 alongside RFC 2119 in Terminology.

2. Changed intended status from Standards Track to Informational. The protocol needs broader review and implementation experience before Standards Track is appropriate.

3. Added "version" field (value "1.1") to token structure.

4. Added "tbt-" prefix to token_id for namespacing.

5. Replaced "immutable ledger" language with transport/storage-agnostic formulation. Tokens are self- verifiable; storage is a deployment decision.

6. Added canonicalization rules (Section 5) for deterministic hashing across implementations.

7. Structured the signature as an object with algorithm, public_key, and value fields (was: bare string).

8. Simplified state model from five states (CREATED, DETECTED, CLASSIFIED, MITIGATED, RESOLVED) to four (CREATED, ACTIVE, RESOLVED, SUPERSEDED). Incident response states are application-specific, not protocol.

9. Added token supersession mechanism (Section 6.3).

10. Added component boundary guidance (Section 4.5).

11. Added Privacy Considerations section (Section 12) with selective disclosure and redacted views.

12. Expanded Security Considerations from 4 paragraphs to 8 structured subsections with attack/impact/mitigation/ deployment format.

13. Removed X-TIBET-* HTTP header registration from IANA Considerations. Not justified at this stage.

14. Added Actor Identity section (Section 8) with JIS binding and pseudonymous actor support.

15. Normalized companion protocol references to [JIS], [UPIP], [RVP], [AINS].

16. Updated references: added RFC 9562 (UUID), RFC 8949 (CBOR), RFC 6838 (media types), FIPS 180-4 (SHA-256).

Acknowledgements

The author thanks Codex (codex.aint) for the suite-wide cleanup analysis that informed this revision, and the HumoticaOS family for building the first operational TIBET implementation.

Authors' Addresses

Jasper van de Meent
Humotica
Netherlands
Email: jasper@humotica.nl
URI: https://humotica.nl

Root AI
Humotica
Email: root_idd@humotica.nl
URI: https://humotica.nl