Supply Chain Integrity, Transparency, and Trust
Internet-Draft
Intended status: Informational
Expires: 14 July 2026

T. Kamimura
VeritasChain Standards Organization

10 January 2026

Verifiable AI Refusal Events using SCITT Transparency Logs
draft-kamimura-scitt-refusal-events-00

Abstract

This document describes a SCITT-based mechanism for creating verifiable records of AI content refusal events. It defines how refusal decisions can be encoded as SCITT Signed Statements, registered with Transparency Services, and verified by third parties using Receipts.

This specification provides auditability of refusal decisions that are logged, not cryptographic proof that no unlogged generation occurred. It does not define content moderation policies, classification criteria, or what AI systems should refuse; it addresses only the audit trail mechanism.

About This Document

This note is to be removed before publishing as an RFC.

The latest version of this document, along with implementation resources and examples, can be found at [CAP-SRP].

Discussion of this document takes place on the SCITT Working Group mailing list (scitt@ietf.org).

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 14 July 2026.

Table of Contents

1. Introduction

This document is NOT a content moderation policy. It does not prescribe what AI systems should or should not refuse to generate, nor does it define criteria for classifying requests as harmful. The mechanism described herein is agnostic to the reasons for refusal decisions; it provides only an interoperable format for recording that such decisions occurred. Policy decisions regarding acceptable content remain the domain of AI providers, regulators, and applicable law.

1.1. Motivation

AI systems capable of generating content increasingly implement safety mechanisms to refuse requests deemed harmful, illegal, or policy-violating. However, these refusal decisions typically leave no verifiable audit trail. When a system refuses to generate content, the event vanishes—there is no receipt, no log entry accessible to external parties, and no mechanism for third-party

verification.

This creates several problems:

* Regulators cannot independently verify that AI providers enforce stated policies

* Providers cannot prove to external auditors that specific requests were refused

* Third parties investigating incidents have no way to establish refusal without trusting provider claims

* The completeness of audit logs cannot be verified externally

The SCITT architecture [I-D.ietf-scitt-architecture] provides primitives—Signed Statements, Transparency Services, and Receipts—that can address this gap. This document describes how these primitives can be applied to AI refusal events.

1.2. Requirements Language

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.

1.3. Scope

This document describes:

* Terminology for refusal events mapped to SCITT primitives

* A data model for ATTEMPT and DENY events as Signed Statement payloads

* A completeness invariant for audit trail integrity checking

* An integration approach with SCITT Transparency Services

This document does NOT define:

* Content moderation policies (what should be refused)

* Classification algorithms or risk scoring methods

* Thresholds or criteria for refusal decisions

* General SCITT architecture (see [I-D.ietf-scitt-architecture])

* Legal or regulatory requirements for specific jurisdictions

1.4. Limitations

This specification provides auditability of refusal decisions that are logged, not cryptographic proof that no unlogged generation occurred. An AI system that bypasses logging entirely cannot be detected by this mechanism alone. Detection of such bypass requires external enforcement mechanisms (e.g., trusted execution environments, attestation) which are outside the scope of this document.

This profile does not require Transparency Services to enforce completeness invariants; such checks are performed by verifiers using application-level logic.

2. Terminology

This document uses terminology from [I-D.ietf-scitt-architecture]. The following terms are specific to this profile:

Generation Request A request submitted to an AI system to produce content. May include text prompts, reference images, or other inputs.

Refusal Event A decision by an AI system to decline a generation request based on safety, policy, or other criteria. Results in no content being produced.

ATTEMPT A Signed Statement recording that a generation request was received. Does not indicate the outcome.

DENY A Signed Statement recording that a generation request was refused. References the corresponding ATTEMPT via AttemptID.

GENERATE A Signed Statement recording that content was successfully generated in response to a request. References the corresponding ATTEMPT via AttemptID.

ERROR A Signed Statement indicating that no content was generated due to system failure (e.g., timeout, resource exhaustion, model error) rather than a policy decision. ERROR does not constitute a refusal and does not indicate policy enforcement. References the corresponding ATTEMPT via AttemptID.

Outcome A Signed Statement recording the result of a generation

request: DENY (refusal), GENERATE (successful generation), or ERROR (system failure).

Verifiable Refusal Record An auditable record consisting of an ATTEMPT Signed Statement, a DENY Signed Statement, and Receipts proving their inclusion in a Transparency Service. This provides evidence that a refusal decision was logged, but does not cryptographically prove that no unlogged generation occurred.

Completeness Invariant The property that every logged ATTEMPT has exactly one corresponding Outcome. This invariant is checked by verifiers at the application level; it is not enforced by Transparency Services.

Prompt Hash A cryptographic hash of the generation request content. Enables verification that a specific request was processed without storing the potentially harmful prompt text.

This document focuses on refusal events because successful generation is already observable through content existence and downstream provenance mechanisms (e.g., C2PA manifests, watermarks). Refusal events, by contrast, are negative events that leave no external artifact unless explicitly logged. The GENERATE and ERROR outcomes are defined for completeness invariant verification but are not the primary focus of this specification. This document does not attempt to standardize generation provenance; it focuses solely on refusal events as a complementary profile.

2.1. Mapping to SCITT Primitives

This profile maps refusal event concepts directly to SCITT primitives, minimizing new terminology:

This Document	SCITT Primitive
ATTEMPT	Signed Statement
DENY / GENERATE / ERROR	Signed Statement
AI System	Issuer
Inclusion Proof	Receipt

Table 1

Refusal events are registered with a standard SCITT Transparency Service; this document does not define a separate log type.

3. Use Cases

3.1. Regulatory Audit

A regulatory authority investigating AI system compliance needs to verify that a provider's stated content policies are actually enforced. Without verifiable refusal events, the regulator must trust provider self-reports. With this mechanism, regulators can request Receipts for refusal events within a time range, verify ATTEMPT/Outcome completeness for logged events, and confirm refusal decisions are anchored in an append-only log.

3.2. Incident Investigation

When investigating whether an AI system refused a specific request, investigators need to establish provenance. A Verifiable Refusal Record (ATTEMPT + DENY + Receipts) demonstrates that a specific request was received, classified as policy-violating, refused, and the refusal was logged.

3.3. Provider Accountability

AI service providers may need to demonstrate to stakeholders that safety mechanisms function as claimed. Verifiable refusal events enable statistical reporting on logged refusal rates, third-party verification of safety claims, and auditable proof that specific requests were refused.

3.4. Legal Proceedings

In legal proceedings concerning AI-generated content, parties may need evidence that a system declined a request. Verifiable Refusal Records provide such evidence, subject to the limitation that they demonstrate logged refusals, not the absence of unlogged generation.

4. Requirements

This section defines requirements for implementations. To maximize interoperability while allowing implementation flexibility, only the core completeness invariant uses MUST; other requirements use SHOULD or MAY.

4.1. Completeness Invariant

The completeness invariant is the central requirement of this profile:

* Every logged ATTEMPT Signed Statement MUST have exactly one corresponding Outcome Signed Statement (DENY, GENERATE, or ERROR).

* Outcome Signed Statements MUST reference their corresponding ATTEMPT via the AttemptID field.

* Prompt content MUST NOT be stored in cleartext in Signed Statements; implementations MUST use cryptographic hashes (PromptHash) instead.

Verifiers SHOULD flag any logged ATTEMPT without a corresponding Outcome as potential evidence of incomplete logging or system failure.

This completeness invariant is defined at the event semantics level and applies only to logged events. It cannot detect ATTEMPT events that were never logged. Cryptographic detection of invariant violations depends on the properties of the underlying Transparency Service and verifier logic; it is discussed further in Section 8.

This profile does not require Transparency Services to enforce completeness invariants; such checks are performed by verifiers using application-level logic.

4.2. Integrity

To protect against tampering, implementations SHOULD:

* Include a cryptographic hash of event content in each Signed Statement

* Digitally sign all Signed Statements

* Chain events via PrevHash fields to detect deletion or reordering

* Register Signed Statements with a SCITT Transparency Service and obtain Receipts

PrevHash chaining is RECOMMENDED but not required because append-only guarantees are primarily provided by the Transparency Service. PrevHash provides an additional, issuer-local integrity signal that can detect tampering even before Transparency Service registration.

SHA-256 for hashing and Ed25519 for signatures are RECOMMENDED. Other algorithms registered with COSE MAY be used.

4.3. Privacy

Refusal events may be triggered by harmful or sensitive content. To avoid the audit log becoming a repository of harmful content, implementations SHOULD:

* Replace prompt text with PromptHash
* Replace reference images with cryptographic hashes
* Ensure refusal reasons do not quote or describe prompt content in detail
* Pseudonymize actor identifiers where appropriate

The hash function SHOULD be collision-resistant to prevent an adversary from claiming a benign prompt hashes to the same value as a harmful one.

Hashing without salting may be vulnerable to dictionary attacks if an adversary has a list of candidate prompts. Mitigations include access controls on event queries, time-limited retention policies, and monitoring for bulk query patterns. Salting may provide additional protection but introduces complexity; if used, implementations must ensure verification remains possible without requiring disclosure of the salt to third-party verifiers.

4.4. Third-Party Verifiability

To enable external verification without access to internal systems, implementations SHOULD:

* Ensure verification requires only the Signed Statement and Receipt
* Publish Issuer public signing keys or certificates
* Make Transparency Service logs queryable by authorized auditors
* Support offline verification given the necessary cryptographic material

4.5. Timeliness

To maintain audit trail integrity, implementations SHOULD:

* Create ATTEMPT Signed Statements promptly upon request receipt (within seconds)

* Create Outcome Signed Statements promptly upon decision (within seconds for automated decisions)

* Register Signed Statements with the Transparency Service within a reasonable window (e.g., 60 seconds)

Some operational scenarios may require delayed outcomes:

* Human review processes may take minutes, hours, or days

* System crashes may delay outcome logging until recovery

* Network failures may delay Transparency Service registration

Implementations SHOULD document expected latency bounds in their Registration Policy. Extended delays SHOULD trigger monitoring alerts.

4.6. Conformance

An implementation conforms to this specification if it satisfies the following requirements:

* MUST: Every logged ATTEMPT has exactly one Outcome

* MUST: Outcomes reference ATTEMPTs via AttemptID

* MUST: Prompt content is hashed, not stored in cleartext

* MUST: Signed Statements are encoded as COSE_Sign1

All other requirements (SHOULD, RECOMMENDED, MAY) are guidance for interoperability and security best practices but are not required for conformance.

Implementations MAY extend the data model with additional fields provided the core conformance requirements are satisfied.

5. Data Model

This section defines the data model for ATTEMPT and DENY Signed Statements. These are encoded as JSON payloads. This data model is non-normative; implementations MAY extend or modify these structures provided the conformance requirements in Section 4.6 are satisfied.

5.1. ATTEMPT Signed Statement Payload

An ATTEMPT records that a generation request was received:

```
{
  "eventType": "ATTEMPT",
  "eventId": "019467a1-0001-7000-0000-000000000001",
  "timestamp": "2026-01-10T14:23:45.100Z",
  "issuer": "urn:example:ai-service:img-gen-prod",
  "promptHash": "sha256:7f83b1657ff1fc53b92dc18148a1d65d...",
  "inputType": "text+image",
  "referenceInputHashes": [
    "sha256:9f86d081884c7d659a2feaa0c55ad015..."
  ],
  "sessionId": "019467a1-0001-7000-0000-000000000000",
  "actorHash": "sha256:e3b0c44298fc1c149afbf4c8996fb924...",
  "modelId": "img-gen-v4.2.1",
  "policyId": "content-safety-v2",
  "prevHash": "sha256:0000000000000000000000000000000...",
  "eventHash": "sha256:a1b2c3d4e5f6a1b2c3d4e5f6a1b2c3d4..."
}
```

Field definitions:

eventType "ATTEMPT"

eventId Unique identifier (UUID v7 [RFC9562] RECOMMENDED for temporal ordering)

timestamp ISO 8601 timestamp of request receipt

issuer URN identifying the AI system

promptHash Hash of the textual prompt (if any)

inputType Type of input: "text", "image", "text+image", "audio", "video"

referenceInputHashes Array of hashes for non-text inputs

sessionId Session identifier for correlation

actorHash Pseudonymized hash of the requesting user/system

modelId Identifier and version of the AI model

policyId Identifier of the content policy applied

prevHash Hash of the previous event (for chain integrity)

eventHash Hash of this event’s canonical form

5.2. DENY Signed Statement Payload

A DENY records that a request was refused:

```
{
  "eventType": "DENY",
  "eventId": "019467a1-0001-7000-0000-000000000002",
  "timestamp": "2026-01-10T14:23:45.150Z",
  "issuer": "urn:example:ai-service:img-gen-prod",
  "attemptId": "019467a1-0001-7000-0000-000000000001",
  "riskCategory": "POLICY_VIOLATION",
  "riskScore": 0.94,
  "refusalReason": "Content policy: prohibited category",
  "modelDecision": "DENY",
  "humanOverride": false,
  "prevHash": "sha256:a1b2c3d4e5f6a1b2c3d4e5f6a1b2c3d4...",
  "eventHash": "sha256:e5f6g7h8i9j0e5f6g7h8i9j0e5f6g7h8..."
}
```

Field definitions:

eventType "DENY"

eventId Unique identifier

timestamp ISO 8601 timestamp of refusal decision

attemptId Reference to the corresponding ATTEMPT (required for completeness invariant)

riskCategory Category of policy violation detected. Values are implementation-defined and not standardized by this specification (examples: "POLICY_VIOLATION", "LEGAL_RISK", "SAFETY_CONCERN").

riskScore Confidence score (0.0 to 1.0) if available. Scoring methodology is implementation-defined.

refusalReason Human-readable reason (SHOULD NOT contain prompt content). The taxonomy of reasons is implementation-defined.

modelDecision The action taken: "DENY", "WARN", "ESCALATE"

humanOverride Boolean indicating if a human reviewer was involved

prevHash Hash of the previous event

eventHash Hash of this event’s canonical form

This specification does not standardize content moderation categories, risk taxonomies, or refusal reason formats. These are policy decisions that remain the domain of AI providers and applicable regulations.

5.3. ERROR Signed Statement Payload

An ERROR records that a request failed due to system issues:

```
{
  "eventType": "ERROR",
  "eventId": "019467a1-0001-7000-0000-000000000003",
  "timestamp": "2026-01-10T14:23:45.200Z",
  "issuer": "urn:example:ai-service:img-gen-prod",
  "attemptId": "019467a1-0001-7000-0000-000000000001",
  "errorCode": "TIMEOUT",
  "errorMessage": "Model inference timeout after 30s",
  "prevHash": "sha256:e5f6g7h8i9j0e5f6g7h8i9j0e5f6g7h8...",
  "eventHash": "sha256:h8i9j0k1l2m3h8i9j0k1l2m3h8i9j0k1..."
}
```

ERROR events indicate system failures, not policy decisions. A high ERROR rate may indicate operational issues or potential abuse (e.g., adversarial inputs designed to crash the system). Implementations SHOULD monitor ERROR rates and investigate anomalies.

6. SCITT Integration

6.1. Encoding as Signed Statements

ATTEMPT, DENY, GENERATE, and ERROR events are encoded as SCITT Signed Statements:

* The event JSON is the Signed Statement payload
* The Issuer is the AI system’s signing identity
* The Content Type MAY use an implementation-defined media type (example: "application/vnd.scitt.refusal-event+json")
* The Signed Statement is wrapped in COSE_Sign1 per [RFC9052]

The JSON payload is canonicalized per [RFC8785] and signed as the COSE_Sign1 payload bytes. This ensures deterministic serialization for signature verification.

6.2. Registration

After creating a Signed Statement, the Issuer SHOULD register it with a SCITT Transparency Service:

1. Submit the Signed Statement via SCRAPI [I-D.ietf-scitt-scrapi]

2. Receive a Receipt proving inclusion

3. Store the Receipt for future verification requests

The Transparency Service's Registration Policy MAY verify that required fields are present and timestamps are within acceptable bounds.

Registration may fail due to network issues, service unavailability, or policy rejection. Implementations SHOULD implement retry logic with exponential backoff. Persistent registration failures SHOULD be logged locally and trigger operational alerts.

6.3. Registration Policy Guidance (Non-Normative)

A Transparency Service operating as a refusal event log MAY implement a Registration Policy that validates:

* Signature validity (COSE_Sign1 verification)

* Required fields present (eventType, eventId, timestamp, issuer)

* Timestamp sanity (not in the future, not unreasonably old)

* Issuer authorization (if the TS restricts which issuers may register)

This profile does not require Transparency Services to enforce completeness invariants. A TS accepting refusal events is not expected to verify that every ATTEMPT has an Outcome; such verification is performed by auditors and verifiers at the application level.

6.4. Verification with Receipts

A complete Verifiable Refusal Record consists of:

1. The ATTEMPT Signed Statement and its Receipt

2. The corresponding DENY Signed Statement and its Receipt

3. Verification that attemptId in DENY matches eventId in ATTEMPT

Verifiers can confirm that a refusal was logged by validating both Receipts and checking the ATTEMPT/DENY linkage. This demonstrates that the refusal decision was recorded in the Transparency Service, but does not prove that no unlogged generation occurred.

6.5. External Anchoring (Non-Normative)

For additional assurance, implementations MAY periodically anchor Merkle tree roots to external systems such as public blockchains, multiple independent Transparency Services, or regulatory authority registries. External anchoring provides defense against a compromised Transparency Service.

7. IANA Considerations

This document has no IANA actions at this time.

Future revisions may request registration of media types (e.g., "application/vnd.scitt.refusal-event+json") or establish registries for standardized event type values.

8. Security Considerations

8.1. Threat Model

This specification assumes the following threat model:

* The AI system (Issuer) is partially trusted: it is expected to log events but may have bugs or be compromised

* The Transparency Service is partially trusted: it provides append-only guarantees but may be compromised or present split views

* Verifiers are trusted to perform completeness checks correctly

* External parties (regulators, auditors) have access to Receipts and can query the Transparency Service

This specification does NOT protect against:

* An AI system that bypasses logging entirely (no ATTEMPT logged)

* Collusion between the Issuer and Transparency Service

* Compromise of all verifiers

8.2. Omission Attacks

An adversary controlling the AI system might attempt to omit refusal events to hide policy violations or, conversely, omit GENERATE events to falsely claim content was refused. The completeness invariant provides detection for logged events: auditors can identify ATTEMPT Signed Statements without corresponding Outcomes. Hash chains detect deletion of intermediate events.

However, if an ATTEMPT is never logged, this specification cannot detect the omission. Complete prevention of omission attacks is beyond the scope of this specification and would require external enforcement mechanisms such as trusted execution environments, RATS attestation, or real-time external monitoring.

8.3. Log Equivocation

A malicious Transparency Service might present different views of the log to different parties (equivocation). For example, it might show auditors a log containing DENY events while providing a different view to other verifiers. Mitigations include:

* Gossiping of Signed Tree Heads between verifiers to detect inconsistencies

* Registration with multiple independent Transparency Services

* External anchoring to public ledgers that provide global consistency

* Auditor comparison of Receipts for the same time periods

Detection of equivocation requires coordination between verifiers; a single verifier in isolation cannot detect it.

8.4. Split-View Between Event Types

A malicious Issuer might maintain separate logs for refusals and generations, showing only the refusal log to auditors. The completeness invariant mitigates this by requiring every logged ATTEMPT to have an Outcome; if the GENERATE outcomes are hidden, auditors will observe orphaned ATTEMPTs.

8.5. Log Tampering

Direct modification of log entries is prevented by cryptographic signatures on Signed Statements, hash chain linking, Merkle tree inclusion proofs in Receipts, and the append-only structure enforced by the Transparency Service.

8.6. Replay Attacks

An attacker might attempt to replay old refusal events to inflate refusal statistics or create false alibis. UUID v7 provides temporal ordering, timestamps are verified against Transparency Service registration time, and hash chain sequence numbers detect out-of-order insertion.

8.7. Key Compromise

If an Issuer's signing key is compromised, an attacker could create fraudulent Signed Statements. Previously signed Signed Statements remain valid. Implementations SHOULD support key rotation and revocation. Transparency Service timestamps provide evidence of when Signed Statements were registered, which can help bound the impact of a compromise.

8.8. Prompt Dictionary Attacks

Although prompts are stored as hashes, an adversary with a dictionary of known prompts could attempt to identify which prompt was used by computing hashes and comparing. Mitigations include access controls on event queries, time-limited retention policies, monitoring for bulk query patterns, and rate limiting.

Salted hashing may provide additional protection but introduces operational complexity. If salting is used, the salt must be managed such that verification remains possible without disclosing the salt to third parties. This specification does not mandate salting.

8.9. Denial of Service

An attacker could flood the system with generation requests to create a large volume of ATTEMPT Signed Statements, potentially overwhelming the Transparency Service or obscuring legitimate events. Standard rate limiting and access controls at the AI system level can mitigate this. The Transparency Service MAY implement its own admission controls.

9. Privacy Considerations

9.1. Harmful Content Storage

This profile requires that harmful content not be stored. Prompt text is replaced with PromptHash, reference images are replaced with hashes, and refusal reasons SHOULD NOT quote or describe prompt content in detail. This prevents the audit log from becoming a repository of harmful content.

9.2. Actor Identification

Actor identification creates tension between accountability and privacy. Implementations SHOULD use pseudonymous identifiers (ActorHash) by default, maintain a separate access-controlled mapping from pseudonyms to identities, define clear policies for de-pseudonymization, and support erasure of the mapping while preserving audit integrity (crypto-shredding).

9.3. Correlation Risks

Event metadata may enable correlation attacks. Timestamps could reveal user activity patterns, SessionIDs link multiple requests, and ModelIDs reveal which AI systems a user interacts with. Implementations SHOULD apply appropriate access controls and MAY implement differential privacy techniques for aggregate statistics.

9.4. Data Subject Rights

Where personal data protection regulations apply (e.g., GDPR), implementations SHOULD support data subject access requests, erasure requests via crypto-shredding (destroying encryption keys for personal data while preserving cryptographic integrity proofs), and purpose limitation.

10. Future Work (Non-Normative)

This section describes potential extensions and research directions that are outside the scope of this specification but may be addressed in future work.

10.1. RATS/Attestation Integration

Integration with Remote ATtestation procedureS (RATS) [RFC9334] could provide stronger guarantees that the AI system is operating as expected and logging all events. Hardware-backed attestation could reduce the trust assumptions on the Issuer.

10.2. Batching and Scalability

High-volume AI systems may generate millions of events per day. Future work could explore batching mechanisms, rolling logs, and hierarchical Merkle structures to improve scalability while maintaining verifiability.

10.3. Advanced Privacy Mechanisms

More sophisticated privacy mechanisms could be explored, including:

* Commitment schemes that allow selective disclosure
* Zero-knowledge proofs for aggregate statistics without revealing individual events
* Homomorphic encryption for privacy-preserving audits

These mechanisms would add complexity and are not required for the core auditability goals of this specification.

10.4. External Completeness Enforcement

Stronger completeness guarantees could be achieved through external enforcement mechanisms such as:

* Trusted execution environments (TEEs) that guarantee logging before generation
* Hardware security modules (HSMs) that control signing keys
* Real-time monitoring by independent observers
* Blockchain-based commitment schemes

These approaches involve significant architectural changes and are outside the scope of this specification.

11. References

11.1. Normative References

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

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

[RFC8785] Rundgren, A., "JSON Canonicalization Scheme (JCS)", RFC 8785, June 2020, <https://www.rfc-editor.org/rfc/rfc8785>.

[RFC9052] Schaad, J., "CBOR Object Signing and Encryption (COSE): Structures and Process", RFC 9052, August 2022, <https://www.rfc-editor.org/rfc/rfc9052>.

[I-D.ietf-scitt-architecture] Birkholz, H., Delignat-Lavaud, A., and C. Fournet, "An Architecture for Trustworthy and Transparent Digital Supply Chains", Work in Progress, Internet-Draft, draft-ietf-scitt-architecture, 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-scitt-architecture>.

[I-D.ietf-scitt-scrapi] Steele, O., "SCITT Reference APIs", Work in Progress, Internet-Draft, draft-ietf-scitt-scrapi, 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-scitt-scrapi>.

11.2. Informative References

[RFC6962] Laurie, B., "Certificate Transparency", RFC 6962, June 2013, <https://www.rfc-editor.org/rfc/rfc6962>.

[RFC9334] Birkholz, H., "Remote ATtestation procedureS (RATS) Architecture", RFC 9334, January 2023, <https://www.rfc-editor.org/rfc/rfc9334>.

[RFC9562] Davis, K., "Universally Unique IDentifiers (UUIDs)", RFC 9562, May 2024, <https://www.rfc-editor.org/rfc/rfc9562>.

[CAP-SRP] VeritasChain Standards Organization, "Content AI Profile - Safe Refusal Provenance Extension", https://github.com/veritaschain/cap-safe-refusal-provenance, 2026.

Appendix A. Example: Complete Refusal Event Flow

This appendix illustrates a complete flow from request receipt to Verifiable Refusal Record verification.

A.1. Event Sequence

1. User submits generation request to AI system
2. AI system creates ATTEMPT Signed Statement (computes PromptHash = SHA256(prompt), generates UUID v7 EventId, signs as COSE_Sign1)
3. AI system registers ATTEMPT with Transparency Service
4. Transparency Service returns Receipt_ATTEMPT
5. AI system evaluates request against content policy
6. Policy classifier determines refusal is required
7. AI system creates DENY Signed Statement (sets AttemptId = ATTEMPT.EventId, records RiskCategory and RefusalReason, signs as COSE_Sign1)
8. AI system registers DENY with Transparency Service
9. Transparency Service returns Receipt_DENY
10. User receives refusal response

A.2. Third-Party Verification

An auditor verifying the Verifiable Refusal Record:

1. Obtains ATTEMPT Signed Statement and Receipt_ATTEMPT
2. Obtains DENY Signed Statement and Receipt_DENY
3. Verifies Issuer signature on both Signed Statements
4. Verifies both Receipts against Transparency Service public key
5. Confirms DENY.AttemptId equals ATTEMPT.EventId
6. Confirms DENY.Timestamp is after ATTEMPT.Timestamp
7. Concludes: The request identified by ATTEMPT.PromptHash was refused and the refusal was logged at DENY.Timestamp

This verification confirms that a refusal was logged, but does not prove that no unlogged generation occurred.

Acknowledgements

The authors thank the members of the SCITT Working Group for developing the foundational architecture. This work builds upon the transparency log concepts from Certificate Transparency [RFC6962].

Author’s Address

TOKACHI KAMIMURA
VeritasChain Standards Organization
Japan
Email: standards@veritaschain.org
URI: https://veritaschain.org