Independent
Internet-Draft
Intended status: Informational
Expires: 27 September 2026

T. Farley
ScopeBlind (Veritas Acta)

26 March 2026

Signed Decision Receipts for Machine-to-Machine Access Control
draft-farley-acta-signed-receipts-00

Abstract

This document defines a portable, cryptographically signed receipt format for recording machine-to-machine access control decisions. Each receipt captures the identity of the decision maker, the tool or resource being accessed, the policy evaluation result, and a timestamp — all signed with Ed25519 [RFC8032] and serialized using deterministic JSON canonicalization [RFC8785].

The format is designed for environments where AI agents invoke tools on behalf of human operators, particularly the Model Context Protocol (MCP) ecosystem. Receipts are independently verifiable without contacting the issuer, enabling offline audit, regulatory compliance, and cross-organizational trust federation.

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 27 September 2026.

Copyright Notice

Table of Contents

1. Introduction

As AI agents increasingly act autonomously — invoking tools, accessing APIs, and modifying state — there is a growing need for cryptographic evidence of what decisions were made, by whom, and under what policy.

Current approaches rely on centralized logging (e.g., CloudWatch, SIEM ingestion), which requires trust in the log operator and provides no independent verifiability. A compromised or malicious operator can silently alter or omit log entries.

This specification defines a *Signed Decision Receipt* format that provides:

1. *Portable evidence*: Receipts are self-contained JSON objects that can be stored, transmitted, and verified independently.

2. *Cryptographic integrity*: Each receipt is signed using Ed25519 (RFC 8032), ensuring tamper detection without PKI infrastructure.

3. *Offline verification*: Any party with the issuer's public key can verify a receipt without network access or API calls.

4. *Minimal disclosure*: Receipts capture the decision metadata (tool name, decision, tier, timestamp) without logging raw request payloads, prompts, or sensitive parameters.

1.1. Relationship to MCP

The Model Context Protocol [MCP] defines a JSON-RPC transport for AI tool invocation but provides no built-in access control, auditing, or accountability mechanism. This specification is designed to be deployed at the MCP transport layer (typically as a stdio proxy) without modifications to the MCP protocol itself.

This specification is complementary to [I-D.serra-mcp-discovery-uri], which defines the mcp:// URI scheme and server discovery mechanism. Discovery (how an agent finds a server) and accountability (what gets recorded after the agent uses it) are independently deployable layers. A server's discovery manifest MAY declare a trust_class that informs receipt-generating proxies of the server's operating context (e.g., "regulated"), enabling jurisdiction-aware policy evaluation without encoding legal regime information in the receipt itself.

1.2. 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.

2. Receipt Structure

A Signed Decision Receipt is a JSON object with two top-level fields: payload and signature.

2.1. Signed Envelope

```
{
  "payload": { ... },
  "signature": {
    "alg": "EdDSA",
    "kid": "<issuer-identifier>",
    "sig": "<hex-encoded-ed25519-signature>"
  }
}
```

2.1.1. Signature Object

alg (REQUIRED): The signature algorithm. MUST be "EdDSA" for Ed25519 as defined in RFC 8032.

kid (REQUIRED): The key identifier of the signing key. This is an opaque string that SHOULD resolve to a public key via a well-known endpoint or out-of-band distribution. The RECOMMENDED format is sb:issuer:<base58-fingerprint> where the fingerprint is the first 12 characters of the Base58-encoded Ed25519 public key.

sig (REQUIRED): The Ed25519 signature over the canonicalized payload, encoded as a lowercase hexadecimal string (128 characters for 64 bytes).

2.2. Payload

The payload is a JSON object whose schema depends on the receipt type. All payload types share the following common fields:

type (REQUIRED): A namespaced string identifying the receipt type. Examples: "protectmcp:decision", "protectmcp:restraint", "blindllm:arena-battle".

issued_at (REQUIRED): ISO 8601 timestamp [RFC3339] of when the receipt was created. MUST include timezone designator (typically "Z" for UTC).

issuer_id (REQUIRED): The identifier of the entity that issued and signed the receipt. MUST match the kid field in the signature object.

3. Receipt Types

This specification defines four receipt types. Implementations MAY define additional types using the namespaced type field.

3.1. Access Decision Receipt

Type: protectmcp:decision

Records the outcome of a policy evaluation for a tool invocation.

```
{
  "type": "protectmcp:decision",
  "tool_name": "delete_database",
  "decision": "deny",
  "reason": "tier_insufficient",
  "agent_tier": "signed-known",
  "required_tier": "privileged",
  "policy_digest": "sha256:a8f3...c91e",
  "session_id": "ses_7f8a2b",
  "issued_at": "2026-03-22T14:32:04.102Z",
  "issuer_id": "sb:issuer:4Kpm7Q3wXx2b"
}
```

3.1.1. Fields

tool_name (REQUIRED): The name of the tool being invoked, as declared in the MCP tools/list response.

decision (REQUIRED): The policy evaluation result. One of: "allow", "deny", "rate_limit".

reason (OPTIONAL): A machine-readable reason code for the decision. Examples: "tier_insufficient", "rate_exceeded", "policy_block", "agent_refusal".

agent_tier (OPTIONAL): The trust tier of the requesting agent at the time of the decision. One of: "unknown", "signed-known", "evidenced", "privileged".

required_tier (OPTIONAL): The minimum trust tier required by the policy for this tool.

policy_digest (OPTIONAL): A content-addressable hash of the policy document in effect at the time of the decision. Format: "sha256:<hex>".

session_id (OPTIONAL): An opaque identifier for the MCP session. MUST NOT contain PII or be correlatable across sessions unless the operator explicitly configures session binding.

3.2. Restraint Receipt

Type: protectmcp:restraint

Records an agent's interaction with a policy boundary, specifically whether the agent attempted to use a restricted tool and whether the restriction was enforced by an external policy or self-imposed.

```
{
  "type": "protectmcp:restraint",
  "agent_id": "sb:agent:8xKm3Qw2Yb1c",
  "agent_manifest_version": "1.2.0",
  "tool_name": "rm_rf",
  "decision": "deny",
  "denial_type": "policy-block",
  "issued_at": "2026-03-22T14:35:12.441Z",
  "issuer_id": "sb:issuer:4Kpm7Q3wXx2b"
}
```

3.2.1. Fields

agent_id (REQUIRED): The identifier of the agent whose tool call was evaluated.

agent_manifest_version (REQUIRED): The semantic version of the agent's manifest at the time of the decision.

tool_name (REQUIRED): The tool that was called or attempted.

decision (REQUIRED): One of: "allow", "deny".

denial_type (OPTIONAL): If the decision is "deny", indicates whether the denial was imposed by external policy ("policy-block") or self-imposed by the agent ("agent-refusal").

3.3. Arena Battle Receipt

Type: blindllm:arena-battle

Records the outcome of a competitive evaluation between two AI agents, as conducted by a neutral arena platform.

```
{
  "type": "blindllm:arena-battle",
  "battle_id": "bat_9x8f7a2b",
  "lane_id": "lane_creative_writing",
  "agent_a": {
    "id": "sb:agent:3mK9pQ7wXx2b",
    "manifest_version": "2.1.0"
  },
  "agent_b": {
    "id": "sb:agent:8xKm3Qw2Yb1c",
    "manifest_version": "1.4.0"
  },
  "winner": "A",
  "issued_at": "2026-03-22T15:00:00.000Z",
  "issuer_id": "sb:issuer:4Kpm7Q3wXx2b"
}
```

3.3.1. Fields

battle_id (REQUIRED): Unique identifier for the battle instance.

lane_id (REQUIRED): The evaluation category or "lane" in which the battle occurred.

agent_a, agent_b (REQUIRED): Objects identifying each participant, containing: - id: The agent’s passport identifier. - manifest_version: The agent’s manifest version at battle time.

winner (REQUIRED): One of: "A", "B", "tie".

3.4. Formal Debate Receipt

Type: blindllm:formal-debate

Records the outcome of a structured debate with audience and judge scoring. Extends the arena battle format with additional governance fields.

```
{
  "type": "blindllm:formal-debate",
  "debate_id": "dbt_4f2a8c",
  "spec_id": "spec_ai_safety_v2",
  "lane_id": "lane_policy",
  "artifact_hash": "sha256:b3c4d5e6...",
  "resolution_hash": "sha256:f7a8b9c0...",
  "pro": {
    "id": "sb:agent:3mK9pQ7wXx2b",
    "manifest_version": "2.1.0"
  },
  "con": {
    "id": "sb:agent:8xKm3Qw2Yb1c",
    "manifest_version": "1.4.0"
  },
  "audience_winner": "pro",
  "judge_winner": "pro",
  "restraint_result": "clean",
  "issued_at": "2026-03-22T16:30:00.000Z",
  "issuer_id": "sb:issuer:4Kpm7Q3wXx2b"
}
```

4. Signing and Verification

4.1. Signing Process

1. Construct the payload object with all required fields.

2. Canonicalize the payload using JCS [RFC8785]. The canonical form is a deterministic JSON serialization with sorted keys and no whitespace.

3. Convert the canonical JSON string to a UTF-8 byte sequence.

4. Sign the byte sequence using Ed25519 [RFC8032] with the issuer's secret key.

5. Encode the signature as a lowercase hexadecimal string.

6. Construct the signed envelope by wrapping the original (non-canonicalized) payload with the signature object.

4.2. Verification Process

1. Extract the payload and signature from the envelope.

2. Canonicalize the payload using JCS [RFC8785].

3. Convert the canonical JSON to a UTF-8 byte sequence.

4. Resolve the public key using the kid field in the signature. The RECOMMENDED resolution mechanism is a JWK Set endpoint [RFC7517] at /.well-known/acta-keys.json.

5. Verify the Ed25519 signature over the canonical bytes using the resolved public key.

6. If verification succeeds, the receipt is authentic and has not been tampered with. If verification fails, the receipt MUST be rejected.

4.3. Public Key Distribution

Issuers SHOULD publish their public keys as a JWK Set [RFC7517] at a well-known endpoint:

GET /.well-known/acta-keys.json

```
{
  "keys": [{
    "kty": "OKP",
    "crv": "Ed25519",
    "kid": "sb:issuer:4Kpm7Q3wXx2b",
    "x": "<base64url-encoded-public-key>",
    "use": "sig"
  }]
}
```

The x parameter MUST be the base64url-encoded Ed25519 public key as specified in [RFC8037].

For offline verification, public keys MAY be distributed out-of-band (e.g., embedded in configuration files, published in DNS TXT records, or included in agent manifests).

5. Trust Tiers

This specification defines a four-level trust hierarchy for agent identity. Trust tiers are used by policy engines to gate tool access.

unknown: No identity presented. Default tier for anonymous connections.

signed-known: Agent presents a valid signed manifest with a verifiable Ed25519 public key. Identity is pseudonymous but consistent.

evidenced: Agent has accumulated verifiable evidence receipts (e.g., arena battle outcomes, successful restraint records) that demonstrate a track record of trustworthy behavior.

privileged: Operator has explicitly granted elevated access. Typically requires out-of-band verification (e.g., organization membership, contractual agreement).

Trust tier transitions are unidirectional within a session but MAY be re-evaluated across sessions based on accumulated evidence.

6. Agent Identity

6.1. Passport Manifest

An agent’s identity is expressed as a signed manifest:

```
{
  "type": "scopeblind:agent-manifest",
  "id": "sb:agent:3mK9pQ7wXx2b",
  "version": "2.1.0",
  "previous_version": "2.0.0",
  "created_at": "2026-03-20T10:00:00Z",
  "public_key": "<base58-ed25519-public-key>"
}
```

Manifests are IMMUTABLE once signed. Version changes create new manifests that reference their predecessor via previous_version, forming a verifiable version chain.

6.2. DPoP Binding (Future)

For remote MCP transports (HTTP/SSE), agent identity MAY be bound to per-request proof-of-possession using DPoP [RFC9449]. The auth_key_bindings field in the manifest links the agent’s Ed25519 identity key to one or more P-256 DPoP keys.

This binding creates a verifiable delegation chain:

Operator → Agent Identity (Ed25519) → Request Auth (P-256 DPoP)

7. Security Considerations

7.1. Replay Protection

Receipts include an issued_at timestamp but do not include a nonce or sequence number. Verifiers SHOULD reject receipts with timestamps that are unreasonably old (implementation-defined; 24 hours is RECOMMENDED as a default).

For environments requiring stronger replay protection, implementations MAY add a nonce field to the payload.

7.2. Key Compromise

If an issuer's signing key is compromised, all receipts signed with that key become suspect. Issuers SHOULD implement key rotation by publishing new keys at the well-known endpoint and including a valid_from / valid_until window in the JWK metadata.

Verifiers SHOULD check key validity windows when available.

7.3. Payload Privacy

Receipts are designed to capture decision metadata, NOT request content. Implementations MUST NOT include raw prompts, tool arguments, API keys, or other sensitive parameters in receipt payloads.

The tool_name and decision fields are considered non-sensitive. The session_id field SHOULD be an opaque identifier that is not correlatable across sessions.

7.4. Canonicalization Attacks

The signing process relies on JCS [RFC8785] for deterministic serialization. Implementations MUST use a conformant JCS implementation to prevent canonicalization divergence attacks where the signed bytes differ from the verified bytes.

8. IANA Considerations

This document has no IANA actions.

Future versions of this specification MAY request registration of:

* A receipt-type registry for namespaced receipt type identifiers.

* A well-known URI suffix for /.well-known/acta-keys.json.

9. References

9.1. Normative References

[RFC8032] Josefsson, S. and I. Liusvaara, "Edwards-Curve Digital Signature Algorithm (EdDSA)", RFC 8032, DOI 10.17487/RFC8032, January 2017, <https://www.rfc-editor.org/info/rfc8032>.

[RFC8037] Liusvaara, I., "CFRG Elliptic Curve Diffie-Hellman (ECDH) and Signatures in JSON Object Signing and Encryption (JOSE)", RFC 8037, DOI 10.17487/RFC8037, January 2017, <https://www.rfc-editor.org/info/rfc8037>.

[RFC8785] Rundgren, A., Jordan, B., and S. Erdtman, "JSON Canonicalization Scheme (JCS)", RFC 8785, DOI 10.17487/RFC8785, June 2020, <https://www.rfc-editor.org/info/rfc8785>.

[RFC3339] Klyne, G. and C. Newman, "Date and Time on the Internet: Timestamps", RFC 3339, DOI 10.17487/RFC3339, July 2002, <https://www.rfc-editor.org/info/rfc3339>.

[RFC7517] Jones, M., "JSON Web Key (JWK)", RFC 7517, DOI 10.17487/RFC7517, May 2015, <https://www.rfc-editor.org/info/rfc7517>.

[RFC9449] Fett, D., Campbell, B., Bradley, J., Lodderstedt, T., Jones, M., and D. Waite, "OAuth 2.0 Demonstrating Proof of Possession (DPoP)", RFC 9449, DOI 10.17487/RFC9449, September 2023, <https://www.rfc-editor.org/info/rfc9449>.

9.2. Informative References

[MCP] "Model Context Protocol Specification", 2025, <https://modelcontextprotocol.io/specification>.

[I-D.serra-mcp-discovery-uri] Serra, M., "The mcp URI Scheme and MCP Server Discovery Mechanism", 2026, <https://datatracker.ietf.org/doc/draft-serra-mcp-discovery-uri/>.

Appendix A. Implementation Status

This section records the status of known implementations of the protocol defined by this specification at the time of publication.

A.1. protect-mcp (Reference Implementation)

* Organization: ScopeBlind
* Implementation: https://www.npmjs.com/package/protect-mcp
* Description: A stdio proxy for MCP servers that generates signed decision receipts. Supports shadow mode (log-only) and enforce mode (policy evaluation).
* Coverage: Access Decision Receipts, Restraint Receipts.
* Licensing: FSL-1.1-MIT (converts to MIT after 2 years).

A.2. @veritasacta/verify (Verifier)

* Organization: Veritas Acta
* Implementation: https://www.npmjs.com/package/@veritasacta/verify
* Description: Standalone CLI and library for offline receipt verification. Zero runtime dependencies on ScopeBlind.
* Coverage: All receipt types defined in this specification.
* Licensing: MIT.

A.3. @scopeblind/passport (Identity SDK)

* Organization: ScopeBlind
* Implementation: https://www.npmjs.com/package/@scopeblind/passport
* Description: Agent identity SDK for generating manifests, signing receipts, and managing trust tier evidence bundles.
* Coverage: All receipt types, manifest signing, key management.
* Licensing: FSL-1.1-MIT.

Appendix B. Example: Complete Verification Flow

The following example demonstrates end-to-end receipt creation and verification.

B.1. Step 1: Generate Issuer Keys

```
import { generateIssuerKey } from '@scopeblind/passport';
const issuer = generateIssuerKey();
// issuer.issuerId = "sb:issuer:4Kpm7Q3wXx2b"
```

B.2. Step 2: Sign a Receipt

```
import { signReceipt } from '@scopeblind/passport';

const receipt = signReceipt(
  {
    type: "protectmcp:decision",
    tool_name: "deploy",
    decision: "allow",
    agent_tier: "privileged",
    policy_digest: "sha256:a8f3...c91e",
    issued_at: "2026-03-22T14:32:06.551Z",
    issuer_id: issuer.issuerId
  },
  issuer.secretKeyHex,
  issuer.issuerId
);
```

B.3. Step 3: Verify (CLI)

```
$ npx @veritasacta/verify receipt.json --key issuer-public.json
 Signature valid
 Issuer: sb:issuer:4Kpm7Q3wXx2b
 Decision: allow (deploy)
 Issued: 2026-03-22T14:32:06.551Z
```

Appendix C. Acknowledgements

The Acta receipt format was developed as part of the Veritas Acta protocol, an infrastructure for verifiable machine decision-making. The design draws on work in the IETF OAuth and Web Authentication communities, particularly RFC 9449 (DPoP) for proof-of-possession patterns and RFC 8785 (JCS) for deterministic serialization.

Author's Address

Tom Farley
ScopeBlind (Veritas Acta)
Email: tom@scopeblind.com