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

T. Farley
ScopeBlind (Veritas Acta)

25 April 2026

Signed Decision Receipts for Machine-to-Machine Access Control
draft-farley-acta-signed-receipts-01

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 October 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. The mandatory-to-implement algorithm is "EdDSA" for Ed25519 as defined in RFC 8032. Implementations MAY additionally support "ES256" (ECDSA with P-256 and SHA-256) to accommodate systems that use P-256 keys natively. Verifiers MUST support "EdDSA" and SHOULD support "ES256".

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.

payload_digest (OPTIONAL): A content-addressable hash of associated

data (tool input, tool output, or other large payloads) that is too large to embed in the receipt. Format: an object containing hash (SHA-256 hex string), size (byte count), and optionally preview (first 256 characters). Enables integrity verification of associated data without embedding it in the receipt.

hook_latency_ms (OPTIONAL): The time in milliseconds spent evaluating the policy decision. Enables operators to verify that security middleware adds negligible overhead (<5ms is RECOMMENDED for synchronous policy checks).

tool_duration_ms (OPTIONAL): The time in milliseconds between the initiation and completion of the tool invocation. Present only in post-execution receipts.

sandbox_state (OPTIONAL): Whether the execution environment had OS-level containment active at the time of the decision. One of: "enabled", "disabled", "unavailable". Enables auditors to verify that agent actions occurred within a sandboxed environment.

action_ref (OPTIONAL): A cross-engine correlation anchor: the SHA-256 hash of the canonical representation of the action being evaluated. When two or more governance engines evaluate the same action independently, both receipts carry the same action_ref, enabling bilateral verification without either engine needing to trust the other.

The computation is:

```
action_ref = SHA-256(canonicalize({
  agentId,
  actionType,
  scopeRequired,
  timestamp
}))
```

where canonicalize follows RFC 8785 (JCS) and scopeRequired is sorted lexicographically before canonicalization. The resulting hex string is deterministic: same inputs produce the same hash regardless of implementation language.

This field is RECOMMENDED when the receipt will be consumed by external governance frameworks or cross-engine audit tools.

verifier_sigil (OPTIONAL): The fingerprint of the Sigil commitment

carried by the verifier that checked this receipt. The Sigil is a deterministic visual artifact derived from the verifier's project public key and a policy containing the verifier's source code hash. This field allows downstream consumers to confirm that a receipt was checked by a known, unmodified verifier.

Format: an 8-character hexadecimal string (the first 8 characters of the Sigil's SHA-256 derivation hash).

This field is produced by the verifier at verification time, not by the receipt issuer at signing time. It is OPTIONAL and informational.

iteration_id (OPTIONAL): A logical iteration grouping identifier for multi-step agent workflows. When an agent runs in an optimization loop or a multi-round deliberation, receipts within the same iteration share the same iteration_id. This enables behavioral analysis tools to group receipts by logical iteration rather than by receipt count.

The value is an opaque string. Hierarchy may be encoded by convention using dot-separated segments (e.g., "run_7.sub_3" encodes a nested iteration at depth 2). The field is agent-declared metadata for grouping purposes, NOT a security boundary — a malicious agent could set misleading values. Receipt signatures cover this field (tamper-evident) but do not vouch for its semantic accuracy.

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. Agent Lifecycle Receipt

Type: protectmcp:lifecycle

Records agent lifecycle events in multi-agent orchestration systems (swarms, coordinator mode, scheduled agents). Enables reconstruction of the full agent topology from the receipt DAG.

```
{
  "type": "protectmcp:lifecycle",
  "lifecycle_event": "subagent_start",
  "agent_id": "worker-a1b",
  "agent_type": "general-purpose",
  "parent_session_id": "ses_7f8a2b",
  "team_name": "backend-ops",
  "sandbox_state": "enabled",
  "issued_at": "2026-04-01T10:15:00.000Z",
  "issuer_id": "sb:issuer:4Kpm7Q3wXx2b"
}
```

3.4.1. Fields

lifecycle_event (REQUIRED): One of: "subagent_start", "subagent_stop", "session_start", "session_end", "task_created", "task_completed", "teammate_idle", "config_change".

agent_id (OPTIONAL): The identifier of the agent involved in the lifecycle event.

agent_type (OPTIONAL): The type of agent (e.g., "coordinator", "worker", "standalone", "general-purpose").

parent_session_id (OPTIONAL): The session identifier of the parent/coordinator agent. Creates a verifiable parent-child relationship in the DAG.

team_name (OPTIONAL): The team or swarm name, if operating in multi-agent mode.

3.5. Spending Authority Receipt

Type: scopeblind:spending_authority

Records the outcome of an issuer-blind spending authorization check. Proves that an agent’s purchase is within authorized limits without revealing organizational identity, budget ceiling, or delegation chain.

```
{
  "type": "scopeblind:spending_authority",
  "amount": 99.50,
  "currency": "USD",
  "category": "cloud_compute",
  "utilization_band": "low",
  "decision": "allow",
  "policy_digest": "sha256:a8f3...c91e",
  "issued_at": "2026-04-01T12:00:00.000Z",
  "issuer_id": "sb:issuer:4Kpm7Q3wXx2b"
}
```

3.5.1. Fields

amount (REQUIRED): The monetary amount of the transaction.

currency (REQUIRED): ISO 4217 currency code (e.g., "USD").

category (OPTIONAL): A classification of the spending type (e.g., "cloud_compute", "saas_subscription", "api_usage").

utilization_band (OPTIONAL): A coarse indicator of budget utilization. One of: "low" (<25%), "medium" (25-75%), "high" (75-100%), "exceeded". The exact budget ceiling is NOT disclosed — only the band — preventing price discrimination while proving budget compliance.

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

The receipt deliberately EXCLUDES: organization name, total budget ceiling, exact remaining budget, agent identity, and delegation chain. Verification uses VOPRF (RFC 9497) to confirm receipt validity without the verifier learning which organization issued it.

3.6. 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. Commitment Mode (Optional Extension)

This section defines an OPTIONAL extension that allows a receipt to carry cryptographic commitments to its field values in place of (or alongside) cleartext. Selective disclosure is performed by revealing inclusion proofs for individual fields. The commitment construction is SHA-256 over a salt-prefixed canonical leaf, organized into a Merkle tree following [RFC6962] domain-separation conventions.

A receipt that uses commitment mode includes a single new field, committed_fields_root, in its payload. Verifiers that do not recognize this field MUST ignore it when performing signature verification, as the field is part of the canonical signed payload but its semantics are scoped to consumers that opt into commitment mode.

The leaf-level commitment construction is byte-identical to that described in [I-D.ietf-oauth-selective-disclosure-jwt] ("SD-JWT"). The Merkle tree organization is byte-identical to Certificate Transparency [RFC6962]. Implementations of this extension SHOULD reuse code from those ecosystems where possible.

5.1. Merkle Tree Construction

Implementations MUST use the following RFC 6962-style construction with explicit one-byte domain separation:

```
leaf_hash     = SHA-256(0x00 || canonical_leaf_bytes)
internal_hash = SHA-256(0x01 || left_child_hash || right_child_hash)
```

For a list of leaf hashes D of length n:

* If n == 1, the Merkle root is D[0].

* Otherwise, let k be the largest power of two strictly less than n. The root is SHA-256(0x01 || MerkleRoot(D[0..k]) || MerkleRoot(D[k..n])).

This construction handles non-power-of-two leaf counts without padding by recursively splitting on the largest power of two, matching [RFC6962] Section 2.1.

The committed_fields_root field MUST be the lowercase hex encoding of the resulting 32-byte Merkle root.

5.2. Canonical Leaf Layout

Each leaf encodes a single committed field as a JCS [RFC8785]-canonicalized JSON object:

```
canonical_leaf_bytes = JCS({
  "name":  field_name,
  "salt":  base64url_unpadded(salt_bytes),
  "value": field_value
})
```

The name field MUST be the field's identifier within the receipt payload (e.g., "principal", "action"). Including the field name in the leaf binds the commitment to a specific field and prevents cross-field substitution attacks. The value field MUST be the cleartext field value, preserving its original JSON type (string, number, boolean, object, or array). The salt field MUST be the base64url-encoded (without padding, per [RFC4648] Section 5) byte string of the salt.

5.3. Leaf Sort Order

Leaves MUST be ordered by the byte-lexicographic order of the UTF-8 encoded name field. Implementations MUST NOT apply locale-aware collation, case folding, or Unicode normalization. Two implementations that disagree on sort order will produce different roots; this rule eliminates that source of interoperability failure.

5.4. Salt Construction and Storage

Each committed field MUST have its own salt. Salts MUST be at least 16 bytes and SHOULD be 32 bytes. Implementations SHOULD generate salts using a cryptographically secure random number generator (e.g., crypto.getRandomValues in browsers, or os.urandom on POSIX).

Implementations MAY derive salts deterministically from a per-subject master secret, but MUST ensure that erasure of one subject's master secret cannot affect commitments belonging to other subjects. The simplest implementation strategy is to use a fresh random salt per field per receipt; this is the RECOMMENDED default.

The salt-prefix ordering for the leaf-level hash inside the canonical leaf JSON object is implementation-internal: the leaf hash is SHA-256 over the JCS-canonicalized object, not over a raw concatenation. This sidesteps salt-ordering ambiguity entirely.

For implementations that emit a separate per-field commitment value alongside the Merkle leaf (for example, in earlier draft versions or in a per-field disclosure envelope), the per-field commitment MUST be computed as SHA-256(salt || value_bytes) where value_bytes is the UTF-8 encoding of the JSON-stringified value. Implementations MUST NOT use the reverse ordering SHA-256(value_bytes || salt); receipts that do so MUST be rejected as malformed.

5.5. Selective Disclosure

To disclose a single field to a verifier, the discloser provides:

1. The cleartext field name, value, and salt.

2. A Merkle inclusion proof, consisting of the leaf's zero-based index within the canonically-sorted leaf list, the total tree_size, and the ordered list of siblings (hex-encoded SHA-256 hashes) along the path from the leaf to the root.

The verifier reconstructs the leaf hash from the disclosed (name, salt, value) tuple, walks the inclusion proof, and compares the resulting root to committed_fields_root in the signed receipt. Mismatch MUST cause the disclosure to be rejected.

A formal Disclosure object with recipient binding, expiry, custody chain, and revocation handle is OUT OF SCOPE for this draft and is deferred to a future revision. Implementations MAY use a minimal disclosure envelope of {name, value, salt, proof} in the interim.

5.6. Signature Scope (Normative Clarification)

The Ed25519 (or other algorithm-agile) signature MUST cover SHA-256(JCS(payload)), where payload is the receipt object with the signature field removed prior to canonicalization. The signature field MUST be removed from the object, not set to null or to the empty string; these produce different JCS output and break interoperability.

5.7. Chain Hash Scope (Normative Clarification)

The previousReceiptHash field MUST be the lowercase hex encoding of SHA-256(JCS(receipt)), where receipt is the entire signed receipt object including the signature field. Including the signature in the chain hash binds the chain to specific signed bytes, so re-signing an identical payload produces a distinct chain link.

5.8. Signature Algorithm Agility

Receipts MAY be signed with any of the following algorithms, identified by their JOSE/JWS algorithm names [RFC7518]:

* EdDSA (Ed25519, [RFC8032]): Mandatory-to-implement (MTI) baseline.

* ML-DSA-65 (FIPS 204): RECOMMENDED for new deployments and post-quantum readiness.

* ES256 (ECDSA over P-256): Permitted for compatibility with existing credentials, particularly those used in [I-D.google-cfrg-libzk] ZK proof composition.

The signature object MUST contain an alg field carrying the algorithm name as a string. Verifiers MUST verify each receipt against the algorithm declared in its own signature.alg field. Chains MAY contain receipts signed under different algorithms; verifiers MUST handle each receipt independently.

The Merkle tree, hash chain, and JCS canonicalization constructions are post-quantum-safe (SHA-256 provides 128-bit security against quantum pre-image and collision attacks). The only classical-cryptography component in this specification is the outer signature, which the algorithm-agility mechanism above allows to upgrade in place.

5.9. Context Binding for ZK Proofs

When a receipt's principal (or any committed field) references a zero-knowledge proof produced under a separate ZK protocol (e.g., [I-D.google-cfrg-libzk]), the ZK proof MUST bind to the receipt's committed_fields_root as a public input or context hash. Without this binding, a valid ZK proof is replayable into a different receipt with a different committed_fields_root.

The committed-fields root serves double duty: it commits to the selectively-disclosable receipt content, and it provides a deterministic, signature-independent context hash that any external proof system can bind against.

5.10. Test Vectors

An interoperability test suite is published alongside this draft. The minimum set is:

* A cleartext receipt with no committed_fields_root field (signature and JCS output only).
* A receipt with four committed fields: the expected Merkle root and an inclusion proof for each field.
* A chain of three receipts: the expected previousReceiptHash values.
* A tampered Merkle proof: MUST fail verification.
* An algorithm-mixed chain (Ed25519 followed by ML-DSA-65): MUST verify successfully when each receipt is checked against its own algorithm.

* A non-power-of-two leaf count (e.g., five committed fields): exercises the recursive split rule.

Test vectors use fixed (non-random) salts to remain reproducible across implementations. Production deployments MUST NOT reuse the test-vector salts and MUST follow the salt construction guidance in Section 5.4.

6. 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.

7. Agent Identity

7.1. 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.

7.2. 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)

8. Security Considerations

8.1. 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.

8.2. 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.

8.3. 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.

8.4. 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.

8.5. Key Distribution and Trust Anchors

Receipts are designed to be verified by parties who do not trust the issuer and did not observe the decision being made. This property (the "issuer-blind" property) is the basis for the third-party auditability claim of this document. The property holds only if the verification key a verifier uses to check a signature is sourced through a channel outside the signed payload itself.

Verifiers MUST NOT accept a verification key transported inside the receipt envelope (including but not limited to fields such as public_key, verification_key, verification_jwk, or any equivalent name under the signed payload) unless that key is independently anchored by an authenticated trust source. An embedded key is strictly controllable by any party able to produce the rest of the payload; a signature that verifies under an attacker-chosen key provides no authenticity guarantee against tampering and is strictly worse than no signature at all, because it provides false assurance.

Conformant verifiers MUST resolve verification keys through one or more of the following external mechanisms, in order of preference when multiple are available:

1. A JWK Set [RFC7517] retrieved from a URL bound to the issuer by an independently authenticated trust root (e.g., a server TLS certificate validated via the Web PKI, or a signed DNS record).

2. A DID Document whose authenticity follows from the DID method’s own verification procedure.

3. A trust anchor configured out-of-band by the verifier operator (e.g., a pinned Ed25519 public key loaded from a verifier configuration file).

4. A well-known endpoint of the issuer’s domain, as described in Section 4.3, provided the domain binding is independently authenticatable.

Verifiers SHOULD expose the externally sourced key provenance to the caller (e.g., via a keySource field in structured output) so that the caller can distinguish a verification anchored by --jwks from one

anchored by a locally configured trust anchor. This supports defense in depth when an operator layers key provenance constraints above signature verification.

Implementations MAY provide a deprecated escape hatch (e.g., a command-line flag) that allows acceptance of embedded keys during a migration window. Such an escape hatch MUST be off by default, MUST emit a visible warning when engaged, and MUST be removed in a subsequent release. See the reference verifier [I-D.verify-reference] for one such implementation.

This section was added in draft-02 in response to a public stress-test of draft-01 that identified the embedded-key pattern as a conformance gap. A negative conformance test vector set ([I-D.agent-governance-testvectors]) exercises the required rejection behavior across representative envelope shapes (flat v1, structured v2, Passport-style, full-JWK-embedded).

9. 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.

10. References

10.1. 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>.

[RFC6962] Laurie, B., Langley, A., and E. Kasper, "Certificate Transparency", RFC 6962, DOI 10.17487/RFC6962, June 2013, <https://www.rfc-editor.org/info/rfc6962>.

[RFC7518] Jones, M., "JSON Web Algorithms (JWA)", RFC 7518, DOI 10.17487/RFC7518, May 2015, <https://www.rfc-editor.org/info/rfc7518>.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <https://www.rfc-editor.org/info/rfc4648>.

10.2. 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/>.

[I-D.google-cfrg-libzk] Google, "The libzk Library for Zero-Knowledge Proofs of Predicates over Existing Credentials", 2026, <https://datatracker.ietf.org/doc/draft-google-cfrg-libzk/>.

[I-D.ietf-oauth-selective-disclosure-jwt] Fett, D., Yasuda, K., and B. Campbell, "Selective Disclosure for JWTs (SD-JWT)", 2026, <https://datatracker.ietf.org/doc/draft-ietf-oauth-selective-disclosure-jwt/>.

[I-D.verify-reference]
Farley, T., "Veritas Acta Reference Verifier (@veritasacta/verify)", 2026, <https://www.npmjs.com/package/@veritasacta/verify>.

[I-D.agent-governance-testvectors]
Farley, T., "Agent Governance Cross-Implementation Test Vectors", 2026, <https://github.com/ScopeBlind/agent-governance-testvectors>.

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: Security gateway for MCP servers with Cedar WASM policy evaluation, Ed25519-signed decision receipts, and Claude Code hook integration. Supports stdio proxy mode, HTTP hook server mode, shadow mode (log-only), and enforce mode.
* Coverage: Access Decision Receipts, Restraint Receipts, Agent Lifecycle Receipts, Spending Authority Receipts.
* Licensing: MIT.
* Version: 0.5.0.

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: Apache-2.0.

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: Apache-2.0.

A.4. protect-mcp-adk (Google ADK)

* Organization: ScopeBlind
* Implementation: https://pypi.org/project/protect-mcp-adk/
* Description: Google Agent Development Kit plugin for Ed25519 receipt signing. BasePlugin subclass, wraps tool calls.
* Coverage: Access Decision Receipts.
* Licensing: MIT.

A.5. scopeblind-pydantic-ai (Pydantic AI)

* Organization: ScopeBlind
* Implementation: https://pypi.org/project/scopeblind-pydantic-ai/
* Description: Pydantic AI Capability for Ed25519 receipt signing. Uses wrap_tool_execute hook for policy evaluation and signing.
* Coverage: Access Decision Receipts.
* Licensing: MIT.

A.6. scopeblind-llamaindex (LlamaIndex)

* Organization: ScopeBlind
* Implementation: https://pypi.org/project/scopeblind-llamaindex/
* Description: LlamaIndex CallbackHandler for FUNCTION_CALL events. Signs tool calls via BaseCallbackHandler, composes with other handlers (Langfuse, Arize, etc.).

* Coverage: Access Decision Receipts.

* Licensing: MIT.

A.7. @scopeblind/vercel-ai (Vercel AI SDK)

* Organization: ScopeBlind

* Implementation: https://www.npmjs.com/package/@scopeblind/vercel-ai

* Description: Hooks into Vercel AI SDK's experimental_onToolCallStart and experimental_onToolCallFinish for per-tool-call receipt signing.

* Coverage: Access Decision Receipts.

* Licensing: MIT.

A.8. @scopeblind/langchain (LangChain + LangGraph)

* Organization: ScopeBlind

* Implementation: https://www.npmjs.com/package/@scopeblind/langchain

* Description: Evidence wrapper for LangChain tool calls and LangGraph node transitions. Produces decision/execution/outcome receipt DAGs.

* Coverage: Access Decision Receipts, graph-level receipts.

* Licensing: MIT.

* Version: 0.2.0.

A.9. Microsoft Agent Governance Toolkit (Enterprise Consumer)

* Organization: Microsoft

* Implementation: https://github.com/microsoft/agent-governance-toolkit

* Description: Consumes receipts via a Cedar policy bridge (PR #667), governed examples for software tool calls (PR #1159) and physical attestation (PR #1168). Not a signing implementation; reads and validates receipts produced by other implementations.

* Coverage: Access Decision Receipts (consumer-only).

* Licensing: MIT.

A.10. AWS Cedar for Agents (WASM Policy Engine)

* Organization: AWS / Cedar Policy

* Implementation: https://github.com/cedar-policy/cedar-for-agents

* Description: WASM bindings for Cedar policy evaluation in JS/TS agent hosts. SchemaGenerator (PR #64, merged) and RequestGenerator (PR #73) enable Cedar-native policy evaluation that produces the policy_digest field referenced in receipts.

* Coverage: Policy evaluation inputs; does not produce receipts directly.

* Licensing: Apache-2.0.

A.11. Sigstore Rekor (Transparency Log Anchor)

* Organization: Linux Foundation / Sigstore

* Implementation: https://rekor.sigstore.dev

* Description: Receipts submitted as DSSE entries to Rekor's public transparency log. Provides an independent temporal anchor: the Rekor inclusion proof and signed entry timestamp prove the receipt existed at a specific time without trusting the receipt operator. Working proof-of-concept submitted (issue #2798).

* Coverage: Temporal anchoring of any receipt type.

* Notes: Uses DSSE envelope with payloadType "application/vnd.scopeblind.receipt+json". Ed25519ph (prehash variant) required for the alternative hashedrekord entry type.

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. Changes from -01

This section summarizes the changes from draft-farley-acta-signed-receipts-01.

action_ref: Added action_ref as an OPTIONAL common payload field (Section 2.2). Provides a normative cross-engine correlation anchor computed as SHA-256 of the JCS-canonicalized action inputs. Enables bilateral verification across independent governance engines evaluating the same action. Motivated by successful composition testing between ScopeBlind Cedar and Agent Passport System delegation engines, where 8 receipts from 2 engines verified against a single action reference.

verifier_sigil: Added verifier_sigil as an OPTIONAL common payload field (Section 2.2). Carries the fingerprint of the Sigil commitment on the verifier that checked the receipt. Enables downstream consumers to distinguish receipts verified by the canonical @veritasacta/verify from those verified by forks or modified copies. Produced at verification time, not at signing time.

iteration_id: Added iteration_id as an OPTIONAL common payload field (Section 2.2). Supports behavioral analysis of multi-step agent workflows by providing a logical grouping identifier. Designed for optimization loops, multi-round deliberations, and meta-agent modification chains. Agent-declared, covered by the receipt signature for tamper evidence but not a security boundary.

ES256 algorithm support: Updated the alg field specification (Section 2.1.1) to permit "ES256" (ECDSA with P-256 and SHA-256) as an additional supported algorithm alongside the mandatory-to-implement "EdDSA" (Ed25519). Motivated by ecosystem providers (compliance risk attestation, multi-attestation envelopes) that use P-256 keys natively.

Transparency log anchoring: Documented Sigstore Rekor as an OPTIONAL temporal anchor for receipts. Receipts wrapped in DSSE envelopes (payloadType "application/vnd.scopeblind.receipt+json") can be submitted to Rekor's public transparency log. The Rekor inclusion proof provides an independent timestamp from the Linux Foundation's append-only log, giving receipts a second verification path beyond the signer's Ed25519 key. Motivated by the need to prevent operators from backdating receipts.

Physical attestation use case: Extended the specification's applicability from software agent tool calls to physical sensor attestation. A cold chain attestation sensor (ATECC608B secure element, SHT40 temp, LIS2DH12 accel, L76K GPS, VEML7700 lux) produces receipts using the same format, same canonicalization, and same verification CLI as software agent receipts. Demonstrates that the receipt format is domain-agnostic: the same envelope carries software decisions and physical-world observations.

SLSA provenance mapping: Documented field-by-field correspondence between receipt payload fields and SLSA v1.0 provenance predicates. When an AI agent builds software and protect-mcp generates a signed receipt chain, the chain constitutes build provenance at SLSA L1/L2. The policy_id and policy_digest fields provide governance metadata that standard SLSA provenance does not carry.

Implementation count: Updated from 3 implementations (protect-mcp, verify, passport) to 11 implementations across 8 frameworks, 4 enterprise ecosystems (Microsoft AGT, AWS Cedar, Linux Foundation Sigstore, OWASP DependencyTrack), and 2 registries (npm, PyPI). Combined monthly downloads exceed 10,000.

Appendix D. 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.

The action_ref normative definition was developed in collaboration with the Agent Passport System project. The verifier_sigil mechanism was motivated by the Sigil visual commitment primitive. The iteration_id field was proposed by contributors to the HyperAgents safety policy discussion. Cross-engine composition testing was performed against receipts from ScopeBlind (Cedar policy), Agent Passport System (delegation scope), and AgentID (identity verification).

The Rekor transparency log integration was informed by feedback from Hayden-IO (Sigstore contributor), who clarified the Ed25519ph requirement for hashedrekord entries and confirmed DSSE as the recommended entry type for Ed25519-signed attestations.

The physical attestation extension was motivated by the Australian Emerging Technology Commercialisation Fund (ETCF) grant program and by discussions on Microsoft Agent Governance Toolkit issue #787 (Physical AI agents: OWASP coverage gap for robotic/actuator systems) with contributions from SINT Protocol (physical constraint enforcement) and the agent-governance-vocabulary project (context_dimensions for physical-world policy attributes).

The SLSA provenance mapping was proposed in collaboration with the SLSA framework specification community (issue #1606).

Author’s Address

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