RCPT PROTOCOL

Receipt Chain Protocol for Trust

A Cryptographic Trust Standard for the Agentic Internet

RCPT Labs

Specification Draft v0.6 — March 2026

hello@rcptprotocol.com | rcptprotocol.com

> *Signed, timestamped, tamper-evident records that any agent can generate and any system can verify.*

This is a working draft. Feedback is welcome at hello@rcptprotocol.com.

Changes from v0.5

This revision makes the following corrections based on technical review:

- **Appendix A model name:** Changed to generic placeholder (`example-llm-v1`). The v0.5 changelog stated this change but the example still contained a specific model name.
- **Revocation receipt example (§4.4):** Restored the required `timestamp` and `agent_id` fields, which were accidentally stripped in v0.5.
- **Numbered list formatting (§5.3, §8.2):** Fixed rendering artifact where numbered steps displayed as "• 1." (bullet + number). Now rendered as proper numbered lists.
- **IETF AIMS citation (§7.7, Appendix B):** Added caveat that draft-klrc-aiagent-auth-00 is an individual submission at -00 and has not been adopted by an IETF working group.
- **Tool object field name (§2.2):** Added explicit `server_uri` field name to the Optional Fields table description. Previously the canonical field name appeared only in Appendix A.
- **Unanchored revocation receipts (§4.4):** Added guidance on how verifiers should treat revocation receipts that have not yet been anchored on-chain.
- **RFC 9591 / FROST reference (§4.2):** Added contextual note explaining the relevance of threshold signing for multi-agent key management. Previously listed in Appendix B without body-text reference.
- **Comparison table column widths (§7):** Fixed layout to prevent column header line-breaking.

Abstract

Autonomous AI agents invoke tools, transfer funds, modify files, call APIs, and make decisions on behalf of humans—often without direct human oversight. How do we know what an agent did, when it did it, why it did it, and whether it can be trusted to do it again?

RCPT Protocol is an open cryptographic standard for verifiable agent action receipts—signed, timestamped, tamper-evident records that any agent can generate and any system can verify. Built on W3C Decentralized Identifiers (DIDs), Ed25519 digital signatures, and JSON Canonicalization Scheme (JCS, RFC 8785), RCPT receipts are the trust primitive the agentic internet requires.

This document specifies the RCPT Protocol data model, cryptographic requirements, receipt chaining semantics, agent identity model, storage architecture, delegation framework, and implementation guidance for developers and enterprises.

Contents

1. The Problem

2. The RCPT Receipt

3. Receipt Chaining

4. Agent Identity

5. Receipt Storage and Anchoring

6. Trust Score Derivation

7. Comparison to Existing Standards

8. Implementation Guidance

9. Security Considerations

10. Privacy Considerations

11. Versioning and Governance

12. Roadmap and Current Status

13. Conclusion

Appendix A. Complete Receipt Example

Appendix B. Normative and Informative References

1. The Problem

1.1 Agents Act Without Accountability

When a human sends an email, signs a contract, or executes a trade, there is an implicit accountability chain. Identity is established. The action is logged. Consequences follow.

When an AI agent does the same things today, there is nothing. No verifiable identity. No tamper-evident log. No standard by which one system can prove to another what happened. The agent acts, and the action dissolves into opacity.

This is not a theoretical problem. It is the reason enterprises cannot deploy agents into regulated workflows. It is the reason compliance officers reject AI automation proposals. It is the reason the promise of autonomous agents remains, for most industries, a promise.

1.2 The MCP Ecosystem Accelerates the Urgency

The Model Context Protocol (MCP) has dramatically lowered the barrier to building and deploying AI agent tools. The number of publicly available MCP servers is growing rapidly. Agents can invoke any of them.

But which ones can be trusted? With what scope? By which agents? Under what conditions? There is no standard answer to any of these questions. Every developer, every enterprise, every AI framework is solving this independently—or not solving it at all.

1.3 The Three Trust Questions

Three questions define the trust gap in agentic AI:

- **Identity:** Who is the agent, and on whose authority does it act?
- **Auditability:** What did it do, and can the record be independently verified?
- **Reputation:** Should it be trusted to do it again, based on verifiable behavioral history?

No existing standard addresses all three cohesively. RCPT Protocol is designed to close this gap.

2. The RCPT Receipt

2.1 What Is a Receipt?

An RCPT receipt is a cryptographically signed, tamper-evident record of a single agent action. It answers the three trust questions simultaneously:

- **Who:** the agent's Decentralized Identifier (DID), encoding its Ed25519 public key.
- **What:** the action taken, the tool invoked, the parameters used, and the outcome.
- **Trust:** the receipt is signed, timestamped, and optionally anchored on-chain—its existence and integrity are independently verifiable.

A receipt is generated by the agent at the moment of action. It is signed with the agent's Ed25519 private key. It is verifiable by any party with access to the agent's public key, which is embedded directly in the DID.

2.2 Receipt Schema

RCPT receipts are JSON documents conforming to the RCPT v0.1 specification. Each receipt contains required fields and optional fields for chaining, delegation, model identification, and anchoring metadata.

> ***Note:*** *This whitepaper documents the v0.1 receipt schema plus proposed extensions (error and revocation action types, tool object, scope as array) targeted for v0.2. The published JSON Schema at github.com/rcpt-protocol covers v0.1. Extensions described here are implemented in the SDKs but not yet formalized in the schema.*

Required Fields

Field	Type	Description
rcpt_version	string	Protocol version ("0.1")
receipt_id	ULID	Unique, lexicographically sortable identifier
timestamp	ISO 8601	UTC with millisecond precision
agent_id	DID	Issuing agent's Decentralized Identifier
action_type	enum	inference | tool_call | transaction | delegation | observation | error* | revocation* | custom
output_hash	sha256:{hex}	SHA-256 hash of the action output
signature	ed25519:{b64url}	Ed25519 signature of the JCS-canonicalized receipt body

** error and revocation are proposed for v0.2. Implementations may use these types; verifiers implementing v0.1 only should treat them as custom.*

Optional Fields

Field	Type	Description
input_hash	sha256:{hex}	SHA-256 hash of the action input
chain	object	Chaining metadata (see Section 3)
delegation	object	Delegation and scope binding (see Section 4.3)
model	object	Model identifier, version, and provider
tool	object	Tool name, version, and server_uri (HTTPS URI of MCP server or tool endpoint)
anchor	object	On-chain anchoring metadata (see Section 5)
metadata	object	Implementation-specific key-value pairs

The **custom** action type serves as a catch-all; implementations using it should adopt a namespaced convention (e.g., "custom:acme.compliance_check") to avoid collisions.

Example Minimal Receipt

```
{
  "rcpt_version": "0.1",
  "receipt_id": "01JQFK8X3YZ4A5B6C7D8E9F0",
  "timestamp": "2026-03-18T14:22:01.342Z",
  "agent_id": "did:key:z6Mkf5rGMoatrSj1f...QLP1RGvMwb5bPCG",
  "action_type": "tool_call",
  "output_hash": "sha256:9f86d081884c7d659a2feaa0c55ad...",
  "signature": "ed25519:3K4lRtBXkFqZ8mN2pY7vW9xA1cD5eH..."
}
```

2.3 Cryptographic Guarantees

- **Authenticity:** the receipt was produced by the agent identified by the DID.
- **Integrity:** the receipt content has not been modified since signing.
- **Non-repudiation:** the signing agent cannot deny having produced the receipt.
- **Temporal proof:** when anchored on-chain, the issuance timestamp is independently verifiable, making retroactive timestamp manipulation detectable.

2.4 Signing Process

The receipt body (all fields except signature and anchor) is serialized to canonical JSON bytes using the JSON Canonicalization Scheme (JCS, RFC 8785). The agent then signs the canonical bytes directly with its Ed25519 private key per RFC 8032. The resulting signature is base64url-encoded and stored in the signature field with the ed25519: prefix.

This design deliberately avoids double-hashing. Ed25519 internally applies SHA-512 during signing; the protocol signs the canonical bytes directly rather than signing a hash of the bytes, in strict compliance with RFC 8032.

2.5 Input and Output Hashing

RCPT receipts do not store raw inputs or outputs—only their SHA-256 hashes. This preserves privacy while enabling verification. A party in possession of the original input can verify it matches the receipt. A party without the input learns only that an action occurred, not its content.

> *RCPT is an audit infrastructure, not a surveillance infrastructure. Receipt content is hash-only by default.*

2.6 Schema Extensibility

Verifiers encountering a receipt with an unknown rcpt_version minor increment (e.g., "0.2" when the verifier implements "0.1") must ignore unrecognized optional fields and proceed with verification of the fields they understand. Unknown required fields must cause verification to fail with an explicit "unsupported version" error. This ensures forward compatibility without silent data loss.

3. Receipt Chaining

3.1 Chain Semantics

Multi-step agent workflows produce sequences of receipts that must be verifiably linked. RCPT receipts support chaining through the optional chain field, which contains:

- **parent_receipt_id:** the receipt ID(s) of the preceding action(s) in the workflow.
- **workflow_id:** a shared identifier grouping all receipts in a workflow.
- **sequence:** the ordinal position of this receipt within the workflow.
- **depth:** the nesting depth for sub-workflows and delegated chains.

The parent_receipt_id field accepts either a single ID (linear chain) or an array of IDs (merge point in a DAG), enabling representation of both sequential and parallel workflow topologies.

3.2 Tamper-Evident Chains

Because each receipt in a chain references its parent by receipt ID, any insertion, deletion, or reordering of receipts within a chain is detectable. A verifier can reconstruct the full chain from the anchored receipt history and identify gaps or inconsistencies.

Combined with on-chain anchoring (Section 5), receipt chains provide a complete, independently verifiable audit trail of multi-step agent workflows—from the initial delegation to the final output.

3.3 Withholding Detection

An agent could attempt to selectively omit receipts from its public history. Receipt chaining mitigates this: because each receipt references its parent, a gap in the chain (a receipt whose declared parent does not exist in the anchored history) is immediately detectable. Verifiers requiring complete audit trails should check for referential integrity across the full chain.

3.4 Cross-Agent Chains

When agent A delegates a sub-task to agent B, agent B's receipts reference agent A's delegation receipt as their chain parent. This creates a cross-agent audit trail: agent A's workflow includes a delegation receipt whose receipt_id appears as the parent_receipt_id in agent B's first receipt. Verifiers can traverse the full chain across agent boundaries by following parent references and resolving each agent's DID independently.

4. Agent Identity

4.1 DID Methods

Every RCPT receipt must be issued by an agent with a verifiable decentralized identity, conforming to the W3C DID Core specification. The recommended DID method for agent identity is did:key, which encodes the agent's Ed25519 public key directly in the identifier:

```
did:key:z6Mkf5rGMoatrSj1f4QLP1RGvMwb5bPCG2gkKvEBSjQdkCas
```

The did:key method requires no registry or resolution infrastructure. The agent's identity is its public key. For enterprise environments where organizational anchoring is needed, did:web is recommended:

```
did:web:company.com:agents:sales-bot-v2
```

Implementations may support additional DID methods (e.g., did:ion, did:sol). The protocol is DID-method-agnostic provided the DID resolves to a valid Ed25519 public key.

4.2 Key Management

Each agent instance must have a unique Ed25519 keypair. Private keys must never leave the agent's secure environment. The protocol requires client-side key generation: the RCPT service acts as a DID resolver, not a key issuer.

- Private keys should be stored in a secure enclave, HSM, or encrypted keystore.
- Key rotation generates a new DID and a delegation receipt linking the old and new identities.
- Compromised keys should be revoked by publishing a revocation receipt (see Section 4.4).
- Implementations should enforce short-lived signing keys with automated rotation.
- For multi-agent deployments where a single logical identity is shared across instances, threshold signing schemes such as FROST (RFC 9591) may be used to distribute signing authority without exposing the full private key to any single instance.

4.3 Delegation and Human Identity Binding

When an agent acts on behalf of a human or organization, the delegation field binds the agent's action to its principal:

```
  "delegation": {
    "delegator_id": "did:web:acme.com:users:jane-smith",
    "scope": ["read:crm", "write:reports"],
    "expires": "2026-03-20T17:30:00.000Z",
    "max_depth": 1
  }
```

This creates an accountability chain: every agent action is traceable to a responsible human or organization, making RCPT receipts legally meaningful in regulated contexts.

> *The **scope** field is an array of strings. Each string is an opaque scope token (e.g., "read:crm"). The v0.1 Python SDK currently emits scope as a comma-separated string for backwards compatibility; the TypeScript SDK emits an array. Both will emit arrays from v0.2 onward. Verifiers should accept either format during the transition period.*

Delegation Chains and Scope Narrowing

When agent A delegates to agent B, which in turn delegates to agent C, each delegation must satisfy two constraints:

- **Scope attenuation (invariant):** A delegatee's scope must be a subset of (or equal to) the delegator's scope. Scope expansion is prohibited and must be flagged as a delegation violation. This is a protocol-level invariant—it is always enforced and is not a per-delegation configuration option.
- **Depth limit:** The optional max_depth field constrains how many levels of re-delegation are permitted. A max_depth of 0 means no re-delegation. If omitted, re-delegation is unlimited.

Delegation Expiry During Workflows

If a delegation expires while a workflow is in progress, the agent must stop generating receipts under that delegation. Any receipt with a timestamp after the delegation's expires field is invalid. Verifiers must reject receipts whose timestamp exceeds the delegation expiry.

4.4 Key Revocation

A key revocation receipt is a receipt of action_type "revocation" signed with the key being revoked:

```
{
  "rcpt_version": "0.1",
  "receipt_id": "01JQFM9R2...",
  "timestamp": "2026-03-19T08:00:00.000Z",
  "agent_id": "did:key:z6Mkf5rG...",
  "action_type": "revocation",
  "output_hash": "sha256:...",
  "revocation": {
    "revoked_did": "did:key:z6Mkf5rG...",
    "reason": "key_compromise",
    "successor_did": "did:key:z6MkN8xP...",
    "effective_after": "2026-03-19T08:00:00.000Z"
  },
  "signature": "ed25519:..."
}
```

The revocation receipt must be anchored on-chain to be discoverable. Verifiers should check for revocation receipts before accepting any receipt signed by that DID. Receipts signed before effective_after remain valid; receipts signed after it using the revoked key are suspect.

An unanchored revocation receipt (one that has been published but not yet settled on-chain) should be treated as advisory: verifiers may flag receipts from the revoked DID as potentially compromised but should not reject them solely on the basis of an unanchored revocation. Once the revocation receipt is anchored, it becomes authoritative and verifiers must enforce it.

5. Receipt Storage and Anchoring

5.1 Storage Architecture

RCPT Protocol separates proof of existence (on-chain) from receipt content (off-chain). The full receipt document is stored in an off-chain receipt store. A Merkle root covering a batch of receipts is periodically anchored to a blockchain, providing a trustless timestamp proof.

This separation follows a Layer 2 pattern: receipts are processed, validated, and batched off-chain, with compressed Merkle roots settled periodically to the base layer. This keeps on-chain costs minimal while ensuring receipts are independently verifiable.

5.2 On-Chain Anchoring

The reference implementation anchors Merkle roots to Solana. The RCPT Anchor Program (Rust, Anchor framework) accepts batched Merkle roots and stores them as immutable Program Derived Accounts (PDAs). Each anchor record contains:

- The Merkle root (32-byte SHA-256 hash)
- The receipt count in the batch
- The anchoring authority's public key
- The on-chain timestamp
- An optional metadata URI

Cost is approximately 320 bytes of on-chain storage per anchor (~0.002 SOL, ~$0.26 at $130/SOL as of March 2026). Any party may anchor their own receipts—no centralized anchoring service is required.

Throughput Targets

The reference implementation targets batches of up to 10,000 receipts per Merkle tree, with anchoring intervals of 5 minutes under normal load. A single anchoring authority can anchor ~2.88 million receipts per day at ~288 transactions (~0.576 SOL/day). Higher throughput is achievable via parallel anchoring authorities or larger batch sizes.

Ledger Requirements

The protocol is not Solana-specific. A conforming ledger must provide: append-only semantics, temporal ordering with verifiable timestamps, public verifiability, and durability for the receipt retention period.

5.3 Merkle Inclusion Proofs

Each anchored receipt includes a Merkle proof. A verifier performs:

1. Fetch the receipt from the off-chain store.
2. Recompute the SHA-256 hash of the JCS-canonicalized receipt body.
3. Walk the Merkle proof to reconstruct the root.
4. Compare the reconstructed root against the on-chain anchor.

A mismatch at any step indicates tampering.

5.4 Availability Requirements

Use Case	Minimum Retention	Guidance
General	7 years	Default for most deployments
Financial services	7 years	Per SEC/FINRA requirements
Healthcare	10–25 years	Per applicable jurisdiction
EU AI Act (high-risk)	Per Article 19	Duration of AI system lifecycle

6. Trust Score Derivation

> ***This section is informational and not normative.*** *Trust scoring is application-specific and subject to gaming, Goodhart's Law, and domain-dependent risk profiles. The weights below are illustrative defaults only. Implementations must calibrate to their own domain.*

6.1 Reference Scoring Algorithm

Signal	Weight	Description
Volume	0.20	Logarithmic scale of total verified receipts (caps at 10,000)
Consistency	0.25	Fraction of actions with complete receipt chains
Age	0.15	Time since first receipt (caps at 365 days)
Verification rate	0.20	Fraction of receipts verified by third parties
Dispute rate	0.20	Inverse of disputed receipt fraction

Score range is 0–100.

6.2 Reference Trust Levels

Score	Level	Description
0–20	Unverified	New or untested agent
21–40	Emerging	Some history, limited verification
41–60	Established	Consistent history, some third-party verification
61–80	Trusted	Strong history, low disputes, regularly verified
81–100	Authority	Extensive history, zero disputes, widely verified

7. Comparison to Existing Standards

7.1 OpenTelemetry

OpenTelemetry is an observability standard for distributed systems, designed for system operators monitoring their own infrastructure. RCPT is designed for verifiable accountability across organizational boundaries—where one party needs to prove to another what an agent did. OpenTelemetry data is mutable and operator-controlled; RCPT receipts are cryptographically signed and independently verifiable. The two are complementary: OpenTelemetry provides operational visibility; RCPT provides evidentiary accountability.

7.2 SPIFFE / SPIRE

SPIFFE defines workload identity for services in distributed systems. RCPT addresses what an autonomous agent did and whether the record can be trusted—across unknown infrastructure boundaries, potentially long after the action occurred. SPIFFE solves identity at invocation time; RCPT solves accountability at audit time.

7.3 X.509 / TLS

X.509 certificates establish identity for static services with known endpoints. AI agents are dynamic—instantiated on demand, running across many hosts, acting on behalf of changing principals. W3C DIDs provide a more flexible identity primitive suited to dynamic agent environments.

7.4 W3C Verifiable Credentials

RCPT receipts are structurally compatible with the VC ecosystem but use a purpose-built schema optimized for agent action semantics. The RCPT data model includes action types, input/output hashing, receipt chaining, and scope declarations that the general VC model does not provide.

7.5 SCITT

IETF SCITT provides an append-only transparency ledger for supply chain claims. RCPT shares SCITT's commitment to append-only semantics and Merkle-based verification, but is purpose-built for AI agent actions. RCPT adds agent identity, receipt chaining, delegation semantics, and trust scoring—none within SCITT's scope.

7.6 Blockchain Audit Logs

General-purpose blockchain audit logging solutions lack standardized schemas for tool invocations, scope declarations, principal accountability chains, or the input/output hashing model that RCPT defines. RCPT provides the semantic layer that generic ledgers do not.

7.7 IETF AIMS

The IETF AIMS draft (draft-klrc-aiagent-auth-00, March 2026) defines a seven-layer composition model for AI agent identity: Identifiers, Credentials, Attestation, Provisioning, Authentication, Authorization, and Monitoring. Its core thesis is that existing standards (WIMSE, SPIFFE, OAuth 2.0, OIDC) compose into a complete agent identity stack without requiring new authentication protocols.

RCPT occupies the gap between AIMS Layer 6 (Authorization) and Layer 7 (Monitoring). AIMS provides no per-action audit trail—authorization stops at the token boundary. RCPT provides cryptographic proof that an authorized action was actually performed, by which agent, at what time, with what inputs and outputs. The two are complementary: AIMS governs who may act; RCPT proves what was done.

> *Note: draft-klrc-aiagent-auth-00 is an individual submission at version -00 and has not been adopted by an IETF working group. Its architecture may change materially. This comparison reflects the draft as published on March 2, 2026.*

Summary Comparison

Capability	RCPT	OTel	SPIFFE	VCs	SCITT	X.509	AIMS
Agent identity	✓	—	✓	✓	—	✓	✓
Action attestation	✓	✓	—	—	✓	—	—
Receipt chaining	✓	Traces	—	—	—	—	—
Crypto signing	✓	—	✓	✓	✓	✓	✓
On-chain anchoring	✓	—	—	—	✓	—	—
Delegation model	✓	—	—	Partial	—	—	Partial
Trust scoring	✓	—	—	—	—	—	—
Cross-org verifiable	✓	—	—	✓	✓	✓	✓

8. Implementation Guidance

8.1 Generating a Receipt

Any agent can generate an RCPT receipt. The minimal implementation requires:

- An Ed25519 keypair and corresponding did:key DID
- A JCS (RFC 8785) canonical serializer
- An Ed25519 signing library (RFC 8032)
- Access to a conforming receipt store (optional: anchoring endpoint)

Reference SDKs are published for Python and TypeScript under Apache 2.0. A Go SDK is planned alongside the hosted API (Phase 3). Framework integrations are available for Anthropic, OpenAI, CrewAI, LangChain, and FastAPI. An MCP server integration (targeting MCP v1.x) enables native receipt generation.

8.2 Verifying a Receipt

1. Fetch the receipt from its storage URL.
2. Check for revocation receipts against the agent's DID.
3. Extract the agent's public key from the DID.
4. JCS-canonicalize the receipt body (all fields except signature and anchor).
5. Verify the Ed25519 signature over the canonical bytes.
6. If delegation is present, verify timestamp is before expiry and scope is authorized.
7. Optionally: query the ledger for anchoring and verify the Merkle inclusion proof.

A receipt that passes signature verification but is not anchored should be treated as unconfirmed.

8.3 Runtime Trust Queries

Agents and orchestration frameworks can query a conforming RCPT trust endpoint before invoking a tool or MCP server. The response contains the entity's trust score, DID, receipt count, revocation status, and a TTL. This enables real-time trust gating without evaluating the full receipt history.

9. Security Considerations

9.1 Key Compromise

If an agent's private key is compromised, an attacker can generate valid receipts on its behalf. Operators should publish a key-revocation receipt (Section 4.4) and rotate to a new keypair immediately. The revocation receipt must be anchored on-chain for discoverability.

9.2 Replay Attacks

A valid receipt from a past action could be replayed to falsely suggest a recent action. RCPT receipts include a precise timestamp and unique ULID. Verifiers must check both and reject receipts outside an acceptable time window.

9.3 Receipt Withholding

An agent could selectively omit receipts to conceal actions. Receipt chaining and on-chain anchoring jointly mitigate this: gaps are detectable via parent reference integrity, and anchored Merkle roots record how many receipts existed at each anchor point.

9.4 Scope Escalation

An agent's scope declaration is a claim, not an enforced constraint. RCPT receipts record actual scope at invocation time. Verifiers should flag receipts where invoked scope exceeds declared scope. Trust scores should penalize repeated violations.

9.5 Delegation Abuse

A malicious agent could attempt to re-delegate beyond permitted scope or depth. Verifiers must enforce the scope attenuation invariant (Section 4.3) and reject delegation chains where a delegatee's scope is not a subset of the delegator's scope. Chains violating max_depth must also be rejected. Delegation receipts expanding scope beyond authorization are invalid regardless of cryptographic validity.

9.6 Algorithm Agility

RCPT uses Ed25519 (128-bit security) for signatures and SHA-256 for hashing. The receipt schema includes explicit algorithm identifiers (ed25519: and sha256: prefixes) to support future migration. Implementations should plan for algorithm agility.

10. Privacy Considerations

10.1 Hash-Only Default

Receipts store SHA-256 hashes of inputs and outputs, never raw data. A minimal receipt reveals only that an agent of a given identity performed an action of a given type at a given time. No content is disclosed.

10.2 GDPR and Right to Erasure

On-chain Merkle roots contain no personal data (only cryptographic hashes). Off-chain receipt content can be deleted or crypto-shredded (encrypting data with a per-record key, then destroying the key) to satisfy erasure obligations while preserving audit trail integrity. Operators in EU jurisdictions should obtain qualified legal guidance.

10.3 Agent Identity Linkability

All receipts from an agent share the same DID, making action history linkable. Implementations may use pairwise DIDs—a different DID per counterparty—to limit linkability while preserving verifiability within each relationship.

11. Versioning and Governance

11.1 Specification Versioning

RCPT Protocol follows semantic versioning (MAJOR.MINOR.PATCH). The current specification is v0.1. Verifiers must support all minor versions within a major version and handle unknown fields per Section 2.6.

11.2 Backwards Compatibility

Minor increments are backwards compatible (new optional fields). Major increments may introduce breaking changes with a minimum six-month migration window.

11.3 Governance

RCPT Protocol v0.x is governed by RCPT Labs. Upon v1.0 stability, stewardship will transfer to an appropriate open standards body. Proposed changes follow an RFC process with 30-day public comment periods.

12. Roadmap and Current Status

The following reflects implementation status as of March 2026.

Phase 1 — Protocol Specification and SDKs (Complete)

RCPT v0.1 specification finalized. JSON Schema published. Python and TypeScript SDKs released under Apache 2.0. CLI tool for receipt generation, verification, and chain inspection. Framework integrations for Anthropic, OpenAI, CrewAI, LangChain, and FastAPI. MCP server integration (targeting MCP v1.x).

Phase 2 — On-Chain Anchoring (Complete)

Solana Anchor Program written and undergoing independent security review. Merkle tree batching and inclusion proof generation implemented in both SDKs. Anchoring available via CLI and programmatic API.

Phase 3 — Hosted API and Trust Scoring (In Progress)

OpenAPI 3.1 specification finalized (22 endpoints, 23 operations). Go implementation underway with Go SDK to follow. Deployment target: api.rcptprotocol.com/v1/.

Phase 4 — Standards Engagement (In Progress)

NIST NCCoE submission prepared. EU AI Act compliance mapping complete (Articles 12, 19, 26). NIST SP 800-207 alignment documented.

Phase 5 — Ecosystem Growth

Public DID resolver. Dashboard for receipt exploration and trust monitoring. Webhook and event surface for real-time receipt streaming. Community governance transition.

13. Conclusion

The agentic internet is being built right now. Agents are being deployed into production workflows without verifiable identity, without audit trails, and without any standard by which their behavior can be evaluated or trusted.

This is the moment to establish the infrastructure that the agentic internet requires—not after the ecosystem has calcified around inferior solutions, but now, while the protocols are still being written and the standards are still being set.

RCPT Protocol is an open contribution to that infrastructure. It is cryptographically sound, built on existing W3C and IETF standards, designed for scale, and implementable today with tools that already exist.

We invite developers, enterprises, AI framework maintainers, and standards researchers to review this specification, implement against it, and contribute to its development.

---

rcptprotocol.com — hello@rcptprotocol.com

Appendix A: Complete Receipt Example

A fully populated RCPT receipt with all optional fields:


```
{
  "rcpt_version": "0.1",
  "receipt_id": "01JQFK8X3YZ4A5B6C7D8E9F0",
  "timestamp": "2026-03-18T14:22:01.342Z",
  "agent_id": "did:key:z6Mkf5rGMoatrSj1f...QLP1RGvMwb5bPCG",
  "action_type": "tool_call",
  "input_hash": "sha256:a3f2b8c1d4e5f6a7b8c9d0e1f2a3b4c5...",
  "output_hash": "sha256:9f86d081884c7d659a2feaa0c55ad015...",
  "chain": {
    "parent_receipt_id": "01JQFK7R1WX3YZ4A5B6C7D8E9",
    "workflow_id": "wf_01JQFK6M0VW2XY3Z4A5B6C7D8",
    "sequence": 3,
    "depth": 0
  },
  "delegation": {
    "delegator_id": "did:web:acme.com:users:jane-smith",
    "scope": ["read:crm", "write:reports"],
    "expires": "2026-03-20T17:30:00.000Z",
    "max_depth": 1
  },
  "model": {
    "provider": "example-provider",
    "name": "example-llm-v1",
    "version": "2026-01-01"
  },
  "tool": {
    "name": "crm_query",
    "version": "2.1.0",
    "server_uri": "https://crm.acme.com/mcp/v1"
  },
  "anchor": {
    "ledger": "solana:mainnet",
    "tx_id": "5KtP...xN2m",
    "merkle_root": "sha256:d4e5f6a7b8c9d0e1...",
    "proof": ["sha256:...", "sha256:...", "sha256:..."]
  },
  "metadata": {
    "latency_ms": 342,
    "token_count": 1847
  },
  "signature": "ed25519:3K4lRtBXkFqZ8mN2pY7vW9xA1cD5eH..."
}
```

Appendix B: Normative and Informative References

Normative

- **RFC 8032:** Edwards-Curve Digital Signature Algorithm (EdDSA)
- **RFC 8785:** JSON Canonicalization Scheme (JCS)
- **W3C DID Core:** Decentralized Identifiers (DIDs) v1.0
- **W3C did:key Method:** did:key Method Specification
- **ULID Specification:** Universally Unique Lexicographically Sortable Identifier
- **ISO 8601:** Date and time format
- **FIPS 180-4:** Secure Hash Standard (SHA-256)

Informative

- **draft-klrc-aiagent-auth-00:** AI Agent Authentication and Authorization (IETF, March 2026). Individual submission; not adopted by a working group as of this writing.
- **draft-ietf-wimse-arch-07:** Workload Identity in a Multi System Environment Architecture (IETF, March 2026)
- **NIST SP 800-207:** Zero Trust Architecture
- **NIST NCCoE:** Accelerating the Adoption of Software and AI Agent Identity and Authorization (February 2026)
- **EU AI Act:** Regulation (EU) 2024/1689
- **RFC 9591:** FROST — Flexible Round-Optimized Schnorr Threshold Signatures. Referenced in Section 4.2 for threshold signing in multi-agent deployments.