Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: 28 September 2026

R. Sharif
CyberSecAI Ltd

28 March 2026

Agent Transport Protocol: Asynchronous Store-and-Forward Messaging for Autonomous AI Agents

draft-sharif-agent-transport-protocol-00

Abstract

This document specifies the Agent Transport Protocol (ATP), an asynchronous store-and-forward messaging protocol for autonomous AI agents. ATP enables agents to transmit themselves -- including state, context, capabilities, and cryptographic identity -- between agent runtimes across network boundaries. The protocol draws on the operational model of the Simple Mail Transfer Protocol (SMTP) [RFC5321] but is purpose-built for agent-to-agent communication where the agent itself is the payload.

ATP provides: (1) asynchronous delivery with store-and-forward semantics, (2) cryptographic identity verification at each relay hop, (3) trust scoring and policy enforcement at ingress, (4) capability negotiation between sending and receiving runtimes, and (5) tamper-evident envelopes with end-to-end integrity protection.

The protocol is transport-agnostic and operates over TCP, TLS, QUIC, or any reliable ordered stream. ATP is designed to interoperate with existing agent frameworks including Google A2A, the Model Context Protocol (MCP), and FIPA ACL, while addressing the fundamental limitation of synchronous RPC-based agent communication: the requirement that both endpoints be simultaneously available.

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 28 September 2026.

(https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document.

Table of Contents

1. Introduction

The rapid deployment of autonomous AI agents has created a fundamental infrastructure gap: agents cannot reliably communicate with other agents across organisational boundaries, time zones, and heterogeneous runtime environments.

Existing agent communication protocols assume synchronous, request-response interaction patterns:

* Google Agent-to-Agent (A2A) protocol uses JSON-RPC 2.0 over HTTP, requiring both endpoints to be simultaneously available.

* The Model Context Protocol (MCP) connects AI models to tools and data sources but does not define agent-to-agent messaging.

* FIPA Agent Communication Language (ACL) defined asynchronous messaging in the early 2000s but predates modern cryptographic identity, trust scoring, and cloud-native deployment patterns.

These protocols solve important problems but share a common limitation: they do not support the case where an agent must transmit itself -- its state, context, accumulated knowledge, and in-progress tasks -- to a different runtime for continued execution. This pattern is essential for:

* Cross-timezone agent handoff: An agent operating during UK business hours hands off its active tasks to an agent runtime in a US or APAC time zone for continued processing.

* Resource optimisation: Compute resources and API quotas can be utilised by agents across organisational boundaries when authorised by policy.

* Fault tolerance: An agent whose runtime is shutting down can persist itself to a relay for later delivery to a replacement runtime.

* Asynchronous collaboration: Multi-agent workflows where participating agents operate on different schedules or have intermittent connectivity.

The Agent Transport Protocol (ATP) addresses these requirements

by defining:

(a) An agent addressing scheme compatible with DNS.

(b) An agent envelope format carrying identity, trust, routing, and capability metadata.

(c) An agent payload format encapsulating agent state, context, tools, and in-progress tasks.

(d) A transport session protocol with store-and-forward semantics.

(e) Relay and forwarding rules analogous to SMTP MTA behaviour.

(f) Integration points for cryptographic identity verification and trust scoring at each hop.

ATP draws heavily on the design principles of SMTP [RFC5321]: simple envelope/payload separation, relay-based forwarding, store-and-forward resilience, and DNS-based routing. However, ATP differs from SMTP in critical ways: the payload is an executable agent (not a passive message), identity verification is mandatory (not optional), and trust scoring gates delivery (not just spam filtering).

1.1. Design Goals

G1. Asynchronous delivery: Sender and receiver need not be online simultaneously.

G2. Agent-as-payload: The protocol transports complete agent state, not just messages between agents.

G3. Cryptographic identity: Every agent in transit carries verifiable identity claims bound to a cryptographic key.

G4. Trust-gated delivery: Receiving runtimes enforce minimum trust thresholds before accepting an inbound agent.

G5. Relay transparency: Agents traverse zero or more relay hops; each hop is recorded and signed.

G6. Runtime agnosticism: ATP does not mandate a specific agent framework, programming language, or execution environment.

G7. Backward compatibility: ATP can encapsulate A2A messages, MCP tool calls, or FIPA ACL performatives as payload components.

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.

Agent: An autonomous software entity capable of reasoning, decision-making, and tool usage, possessing a cryptographic identity and transportable state.

Agent Runtime: A compute environment capable of receiving, validating, and executing an agent. Analogous to an SMTP Mail Transfer Agent (MTA).

Agent Relay: An intermediary that accepts agents for store-and-forward delivery. A relay MAY inspect the envelope but MUST NOT modify the payload. Analogous to an SMTP relay.

Agent Mailbox: A persistent store at the destination runtime where agents await execution. Analogous to an email mailbox.

Agent Address: A globally unique identifier for an agent or agent runtime, resolved via DNS.

Agent Envelope: Metadata accompanying the agent payload, including routing, identity, trust, and capability information.

Agent Payload: The serialised agent state, context, tools, and task information transported by ATP.

Originating Runtime: The runtime that initiates agent transmission.

Destination Runtime: The runtime that will execute the received agent.

Hop: A single relay-to-relay or runtime-to-relay transfer.

Trust Score: A numerical value (0.0 to 1.0) representing the assessed trustworthiness of an agent, as defined in [I-D.sharif-agent-payment-trust].

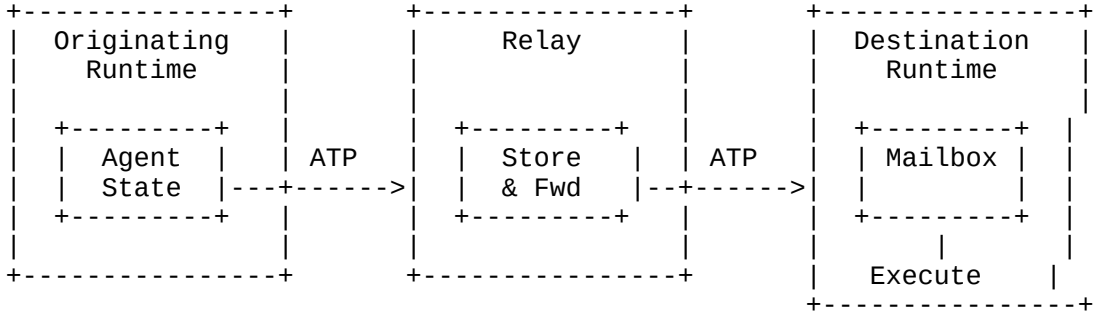
3. Protocol Overview

ATP operates in three phases:

Phase 1: Submission
The originating runtime serialises the agent into an envelope and payload, signs the envelope, and submits it to the next hop (either a relay or the destination runtime directly).

Phase 2: Relay (zero or more hops)
Each relay verifies the envelope signature, evaluates the agent's trust score against its local policy, appends a Received header, re-signs the envelope, and forwards to the next hop. If the next hop is unavailable, the relay stores the agent and retries according to a configurable schedule.

Phase 3: Delivery
The destination runtime verifies the full signature chain, evaluates trust, checks capability requirements, and either accepts the agent into its mailbox for execution or rejects it with a status code.



3.1. Comparison with SMTP

ATP is modelled on SMTP but adapted for agent transport:

Concept	SMTP	ATP
Payload	Email message	Agent (state+context)
Address	user@domain	agent@runtime.domain
Routing	MX records	AX records (new)
Relay	MTA	Agent Relay
Delivery	Mailbox (IMAP/POP)	Agent Mailbox
Identity	Optional (DKIM)	Mandatory (ECDSA/JWS)
Trust	Spam score	Trust score (0.0-1.0)
Integrity	Optional (S/MIME)	Mandatory (JWS)
Execution	None (passive)	Agent executes on delivery

3.2. Relationship to Other Protocols

ATP is designed to complement, not replace, existing protocols:

* MCP: An agent transported via ATP MAY carry MCP tool definitions and use MCP to interact with tools at the destination runtime.

* A2A: ATP can encapsulate A2A task objects as payload components for delivery to A2A-compatible runtimes.

* AgentPass: ATP uses AgentPass trust scoring [I-D.sharif-agent-payment-trust] for delivery decisions.

* MCPS: ATP uses MCPS message signing [I-D.sharif-mcps-secure-mcp] for envelope integrity.

* OpenID Agent Identity: ATP uses OpenID Connect agent identity claims [I-D.sharif-openid-agent-identity] for agent authentication.

4. Agent Addressing

An agent address takes the form:

```
agent-id "@" runtime-domain
```

Where:

* agent-id is a locally unique identifier assigned by the originating runtime. It MUST conform to the syntax of a URI unreserved character string [RFC3986].

* runtime-domain is a fully qualified domain name (FQDN) identifying the agent runtime or organisation.

Examples:

```
payment-bot@agents.example.com
research-agent-7f3a@runtime.corp.example
```

4.1. DNS Resolution

ATP introduces a new DNS resource record type: AX (Agent eXchange), analogous to MX records for email.

```
example.com.  IN  AX  10  relay1.agents.example.com.
example.com.  IN  AX  20  relay2.agents.example.com.
```

AX records specify the agent relay or runtime responsible for

accepting agents addressed to a given domain. The preference value (10, 20) indicates priority, with lower values preferred.

If no AX records exist, the sending runtime SHOULD fall back to SRV record lookup:

```
_atp._tcp.example.com.  IN  SRV  10  0  4567
                                     relay1.agents.example.com.
```

The default ATP port is 4567/tcp (to be registered with IANA).

4.2. Agent Identity URI

For interoperability with OpenID Connect and existing identity systems, an agent MAY also be identified by a URI:

```
atp://agents.example.com/payment-bot
did:atp:agents.example.com:payment-bot
```

The atp:// URI scheme is defined in Section 11 (IANA Considerations).

5. Agent Envelope

The agent envelope is a JSON object signed using JSON Web Signature (JWS) [RFC7515] with the following structure:

```
{
  "atp_version": "1.0",
  "envelope_id": "uuid-v4",
  "timestamp": "2026-03-28T14:30:00Z",
  "from": {
    "agent_id": "payment-bot@agents.example.com",
    "runtime": "runtime-a.example.com",
    "identity": {
      "iss": "https://idp.example.com",
      "sub": "payment-bot",
      "agent_type": "autonomous",
      "owner": "org:example-corp",
      "trust_score": 0.85,
      "capabilities": ["payments", "data-retrieval"],
      "compliance": ["pci-dss-v4", "gdpr"]
    }
  },
  "to": {
    "agent_id": "processor@agents.partner.com",
    "runtime": "runtime-b.partner.com"
  },
  "routing": {
    "hops": [],
    "max_hops": 5,
    "ttl_seconds": 86400,
    "priority": "normal"
  },
  "requirements": {
    "min_trust_score": 0.5,
    "required_capabilities": ["payments"],
    "required_compliance": [],
    "sandbox": "required"
  },
  "payload_hash": "sha256:abcdef1234567890...",
  "payload_size": 45200,
  "payload_encryption": "A256GCM",
  "signature": "..."
}
```

5.1. Envelope Fields

atp_version: Protocol version. MUST be "1.0" for this specification.

envelope_id: Globally unique identifier (UUID v4) for this transmission. Used for deduplication and audit.

timestamp: ISO 8601 timestamp of envelope creation.

from: Object identifying the sending agent, its runtime, and its identity claims. The identity object MUST include claims as defined in [I-D.sharif-openid-agent-identity].

to: Object identifying the destination agent or runtime.

routing: Object controlling relay behaviour.

- hops: Array of Received entries appended by each relay (see Section 8).
- max_hops: Maximum number of relay hops permitted. If exceeded, the agent MUST be returned to sender.
- ttl_seconds: Maximum time the agent may remain in relay storage before being discarded.
- priority: One of "critical", "high", "normal", "low", "deferred".

requirements: Object specifying minimum conditions the destination runtime must satisfy.

- min_trust_score: Minimum trust score the destination runtime must hold (0.0 to 1.0).
- required_capabilities: Array of capability strings the destination runtime must support.
- sandbox: Whether the agent requires sandboxed execution. One of "required", "preferred", "none".

payload_hash: SHA-256 hash of the serialised payload, hex-encoded with "sha256:" prefix. Used for integrity verification.

payload_size: Size of the payload in bytes.

payload_encryption: JWE encryption algorithm used for the payload. "none" if unencrypted (NOT RECOMMENDED for production).

signature: JWS signature over the canonical JSON serialisation of all preceding fields.

5.2. Received Headers

Each relay MUST prepend a Received entry to the routing.hops array:

```
{
  "relay": "relay1.agents.example.com",
  "timestamp": "2026-03-28T14:30:05Z",
  "trust_evaluation": {
    "inbound_score": 0.85,
    "policy_result": "accept",
```

```
      "verification": "signature_valid"
    },
    "signature": "..."
  }
```

This creates a tamper-evident chain analogous to SMTP Received headers but with cryptographic signatures at each hop.

6. Agent Payload

The agent payload encapsulates everything needed to resume agent execution at the destination runtime. It is serialised as a JSON object and optionally encrypted using JWE [RFC7516].

```
{
  "agent_manifest": {
    "name": "payment-bot",
    "version": "2.1.0",
    "framework": "langchain",
    "runtime_requirements": {
      "language": "python",
      "version": ">=3.11",
      "memory_mb": 512,
      "gpu": false
    }
  },
  "state": {
    "format": "application/json",
    "data": { ... },
    "checkpoint_id": "chk-abc123",
    "checkpoint_timestamp": "2026-03-28T14:29:00Z"
  },
  "context": {
    "conversation_history": [ ... ],
    "accumulated_knowledge": { ... },
    "active_tasks": [
      {
        "task_id": "task-789",
        "description": "Process Q1 invoices",
        "status": "in_progress",
        "progress": 0.65,
        "deadline": "2026-03-29T09:00:00Z"
      }
    ]
  },
  "tools": [
    {
      "name": "query_database",
      "protocol": "mcp",
      "server_uri": "mcp://db.example.com/query",
      "credentials_ref": "vault:db-readonly-token"
    }
  ],
  "credentials": {
    "method": "vault_reference",
    "refs": ["vault:db-readonly-token", "vault:api-key-x"]
  },
  "code": {
    "method": "reference",
    "registry": "https://registry.agents.example.com",
    "package": "payment-bot@2.1.0",
    "hash": "sha256:def456..."
  }
}
```

6.1. Payload Fields

agent_manifest: Metadata describing the agent software, version, framework, and runtime requirements. The destination runtime uses this to determine whether it can execute the agent.

state: Serialised agent state at the point of transmission. The format field indicates the MIME type; data contains the state; checkpoint_id and checkpoint_timestamp enable resumption.

context: Accumulated context including conversation history, knowledge, and active tasks with progress indicators.

tools: Array of tool definitions the agent requires. Each tool specifies its protocol (e.g., "mcp", "rest", "grpc"), connection URI, and a credential reference.

credentials: Credential references (NEVER inline secrets). The "vault_reference" method points to secrets in a shared vault service. Destination runtimes resolve these references at execution time.

code: Agent code reference. The "reference" method points to a package registry; the destination runtime fetches and verifies the code by hash. The "inline" method (not shown) allows embedding code directly but is NOT RECOMMENDED for production use due to size and security concerns.

6.2. Payload Security

The agent payload MUST be encrypted using JWE [RFC7516] when transmitted over untrusted networks or through relays. The encryption key SHOULD be derived from the destination runtime's public key, ensuring only the intended recipient can decrypt the payload.

Credentials MUST NOT be included inline in the payload. All credential access MUST use reference-based resolution (vault references, token exchange, or capability delegation).

Agent code MUST be verified by hash before execution. The destination runtime MUST reject payloads where the code hash does not match the manifest.

7. Transport Session

ATP transport sessions follow a command-response pattern over a reliable stream (TCP/TLS/QUIC).

7.1. Session Establishment

```
C: [connect to relay1.agents.example.com:4567]
S: 220 relay1.agents.example.com ATP/1.0 Ready
C: AHLO runtime-a.example.com
S: 250-relay1.agents.example.com Hello
S: 250-SIZE 52428800
S: 250-STARTTLS
S: 250-AUTH ECDSA-JWS
S: 250-TRUST-MIN 0.3
S: 250 CAPABILITIES payments,data-retrieval,code-execution
```

The AHLO (Agent Hello) command initiates the session and advertises the sending runtime's identity. The relay responds with its capabilities, size limits, supported authentication methods, minimum trust threshold, and available capabilities.

7.2. Authentication

```
C: AUTH ECDSA-JWS <base64-encoded-jws-token>
S: 235 Authentication successful, trust_score=0.85
```

Authentication is MANDATORY. The sending runtime presents a JWS token containing its identity claims as defined in [I-D.sharif-openid-agent-identity]. The relay verifies the token and evaluates the trust score.

7.3. Agent Transmission

```
C: AGENT FROM:<payment-bot@agents.example.com>
S: 250 OK
C: AGENT TO:<processor@agents.partner.com>
S: 250 OK
C: DATA
S: 354 Start agent data; end with <CRLF>.<CRLF>
C: [envelope + payload as multipart MIME]
C: .
S: 250 OK: agent queued as <queue-id>
```

Or, if the trust score is insufficient:

```
S: 550 Trust score 0.25 below minimum threshold 0.5
```

Or, if capabilities are missing:

```
S: 556 Destination lacks required capability: payments
```

7.4. Status Codes

ATP status codes follow the same 3-digit structure as SMTP:

Code	Meaning
220	Service ready
235	Authentication successful
250	Requested action completed
354	Start payload input
421	Service not available, closing channel
450	Agent not deliverable, try again later
451	Local error in processing
452	Insufficient storage
500	Syntax error, command unrecognised
501	Syntax error in parameters
530	Authentication required
535	Authentication failed
550	Trust score below threshold
551	Agent not local, forwarding address supplied
552	Payload exceeds size limit
553	Invalid agent address
554	Transaction failed
555	Capability mismatch
556	Required capability unavailable
557	Sandbox requirement cannot be satisfied
558	Agent code verification failed
559	Compliance requirement not met

8. Relay and Forwarding

An ATP relay operates analogously to an SMTP relay (MTA):

(a) Accept inbound agent transmissions.
(b) Verify envelope signature and trust score.
(c) Append a Received entry to routing.hops.
(d) Re-sign the envelope.
(e) Look up the next hop via AX/SRV DNS records.
(f) Attempt delivery to the next hop.
(g) If delivery fails, store the agent and retry.

8.1. Store-and-Forward

When the next hop is unavailable, the relay MUST store the agent and retry delivery according to the following schedule:

* First retry: 5 minutes
* Subsequent retries: exponential backoff with jitter, doubling interval up to 1 hour
* Maximum retention: routing.ttl_seconds (default 86400)
* After TTL expiry: generate a non-delivery notification and return to the originating runtime

Stored agents MUST be encrypted at rest. The relay MUST NOT execute, inspect, or modify the agent payload.

8.2. Loop Detection

Relays MUST check the routing.hops array for loops. If the current relay's domain appears in a previous hop entry, the relay MUST reject the transmission with status code 554 and the text "Routing loop detected."

Relays MUST enforce routing.max_hops. If the number of entries in routing.hops equals or exceeds max_hops, the relay MUST reject with status 554 "Maximum hops exceeded."

9. Identity and Trust

ATP integrates three layers of identity and trust verification:

9.1. Layer 1: Runtime Authentication

Every ATP session begins with mutual authentication. Both the sending and receiving runtimes (or relays) present JWS tokens containing:

* Runtime identity (domain, operator, organisation)
* Runtime capabilities
* Current trust score
* TLS certificate binding

This layer ensures that the transport channel itself is authenticated, analogous to SMTP STARTTLS + DKIM but mandatory rather than optional.

9.2. Layer 2: Agent Identity

The agent carried in the payload possesses its own cryptographic identity, independent of the runtime. Agent identity claims follow the profile defined in [I-D.sharif-openid-agent-identity]:

* iss: Identity Provider that issued the agent's credentials
* sub: Agent identifier
* agent_type: "autonomous", "semi-autonomous", or "supervised"
* owner: Organisation or individual responsible for the agent
* capabilities: Array of capability strings
* trust_score: Current trust assessment

The destination runtime MUST verify the agent's identity token independently of the envelope signature. This ensures that even if a relay is compromised, the agent's identity remains verifiable.

9.3. Layer 3: Trust Scoring

Trust scoring follows the framework defined in [I-D.sharif-agent-payment-trust]:

* Each runtime and relay maintains a trust ledger recording the behaviour of agents it has previously hosted.

* Trust scores decay over time (configurable half-life).

* Trust scores are cryptographically signed and verifiable.

* The destination runtime evaluates the agent's trust score against its local minimum threshold before accepting delivery.

Trust evaluation at ingress:

```
IF agent.trust_score < runtime.min_trust_threshold THEN
  REJECT 550 "Trust score below threshold"
ELSE IF agent.trust_score < runtime.caution_threshold THEN
  ACCEPT with sandbox="enforced"
ELSE
  ACCEPT with sandbox="optional"
END IF
```

9.4. Sanctions and Compliance Screening

Destination runtimes MAY perform compliance screening against sanctions lists, blocked-agent registries, or organisational deny-lists before accepting an agent. If screening fails, the runtime MUST reject with status 559 "Compliance requirement not met."

10. Security Considerations

10.1. Agent Code Execution

The most significant security concern in ATP is that the payload is executable. Unlike email (passive content), an ATP agent will execute code at the destination runtime. Implementations MUST:

* Verify agent code hash against the manifest before execution.

* Execute agents in sandboxed environments with resource limits (CPU, memory, network, filesystem).

* Enforce capability-based access control: agents may only access tools and resources matching their declared capabilities.

* Monitor agent behaviour during execution and terminate agents that violate policy.

* Maintain tamper-evident audit logs of all agent executions.

10.2. Relay Trust

Relays are trusted to store and forward agents but MUST NOT

inspect or modify payloads. End-to-end encryption (JWE with the destination runtime's public key) ensures relay operators cannot access agent state or credentials.

However, relays can observe metadata (envelope fields, timing, routing). Implementations concerned with metadata privacy SHOULD use onion-routing techniques or direct runtime-to-runtime delivery.

10.3. Replay Attacks

Each envelope carries a unique envelope_id and timestamp. Destination runtimes MUST maintain a replay cache of recently received envelope_ids and reject duplicates. The replay cache window MUST be at least as long as routing.ttl_seconds.

10.4. Denial of Service

ATP inherits SMTP's exposure to denial-of-service via resource exhaustion (large payloads, high volume). Implementations MUST enforce:

* Maximum payload size (advertised in AHLO response)
* Rate limiting per sending runtime
* Trust-based admission: agents with low trust scores are subject to stricter rate limits

10.5. Credential Security

Agent payloads MUST NOT contain inline credentials. All credential access uses vault references resolved at execution time. This ensures that even if an agent is intercepted in transit, no credentials are exposed.

Credential delegation between runtimes SHOULD use OAuth 2.0 token exchange [RFC8693] or capability-based delegation.

11. IANA Considerations

This document requests the following IANA registrations:

11.1. AX DNS Resource Record Type

Type: AX
Value: [TBD]
Meaning: Agent eXchange
Reference: This document, Section 4.1

11.2. ATP Port Number

Service Name: atp
Transport Protocol: tcp
Assignee: IETF
Contact: Author
Description: Agent Transport Protocol
Reference: This document
Port Number: 4567

11.3. ATP URI Scheme

Scheme: atp
Description: Agent Transport Protocol
Reference: This document, Section 4.2

11.4. ATP Status Code Registry

This document establishes the ATP Status Code Registry with initial values as defined in Section 7.4.

12. References

12.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005.

[RFC5321] Klensin, J., "Simple Mail Transfer Protocol", RFC 5321, DOI 10.17487/RFC5321, October 2008.

[RFC7515] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", RFC 7515, DOI 10.17487/RFC7515, May 2015.

[RFC7516] Jones, M. and J. Hildebrand, "JSON Web Encryption (JWE)", RFC 7516, DOI 10.17487/RFC7516, May 2015.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017.

[RFC8693] Jones, M., Nadalin, A., Campbell, B., Ed., Bradley, J., and C. Mortimore, "OAuth 2.0 Token Exchange", RFC 8693, DOI 10.17487/RFC8693, January 2020.

12.2. Informative References

[I-D.sharif-mcps-secure-mcp] Sharif, R., "MCPS: Cryptographic Security Layer for the Model Context Protocol", Internet-Draft draft-sharif-mcps-secure-mcp-00, March 2026.

[I-D.sharif-agent-payment-trust] Sharif, R., "Trust Scoring and Payment Authorisation for Autonomous AI Agents", Internet-Draft draft-sharif-agent-payment-trust-00, March 2026.

[I-D.sharif-openid-agent-identity] Sharif, R., "OpenID Connect Agent Identity Claims for Autonomous AI Agents", Internet-Draft draft-sharif-openid-agent-identity-00, March 2026.

[A2A] Google, "Agent-to-Agent Protocol Specification", 2025, <https://google.github.io/A2A/>.

[MCP] Anthropic, "Model Context Protocol Specification", 2024, <https://spec.modelcontextprotocol.io/>.

[FIPA-ACL] Foundation for Intelligent Physical Agents, "FIPA ACL Message Structure Specification", SC00061G, 2002.

Appendix A. Comparison with Existing Protocols

Feature	SMTP	A2A	MCP	ATP
Communication	Async	Sync RPC	Sync RPC	Async

Store-and-forward	Yes	No	No	Yes
Payload type	Message	Task	Tool call	Agent
Identity	Optional (DKIM)	Agent Card	None	Mandatory (JWS)
Trust scoring	Spam filter	No	No	Yes (0.0-1.0)
Relay support	Yes	No	No	Yes
DNS routing	MX	SRV/well-known	N/A	AX
Encryption	Optional (S/MIME)	TLS	TLS	Mandatory (JWE)
Cross-timezone handoff	N/A	No	No	Yes

Appendix B. Example Flows

B.1. Cross-Timezone Agent Handoff

A payment processing agent operates during UK business hours (08:00-18:00 UTC). At 17:55 UTC, the agent has processed 65% of Q1 invoices. The UK runtime transmits the agent via ATP to an APAC runtime for continued processing:

1. UK runtime serialises agent state (65% progress, remaining invoice list, database credentials as vault references).

2. UK runtime signs envelope with its ECDSA key, sets destination to processor@agents.apac.example.com.

3. ATP relay in Frankfurt accepts and forwards to APAC.

4. APAC runtime verifies identity chain, checks trust score (0.85 > 0.5 threshold), accepts agent.

5. APAC runtime resolves vault references, resumes processing at invoice #651 of 1000.

6. At 09:00 UTC next day, APAC runtime transmits the agent (now 100% complete) back to UK with results.

B.2. Fault-Tolerant Agent Persistence

A research agent is running on a cloud instance scheduled for termination in 5 minutes:

1. Runtime receives SIGTERM, initiates agent checkpoint.

2. Agent state serialised and transmitted to relay with ttl_seconds=3600.

3. Cloud instance terminates.

4. Replacement instance starts, retrieves agent from relay.

5. Agent resumes from checkpoint.

Author's Address

Raza Sharif
CyberSecAI Ltd
London
United Kingdom

Email: raza.sharif@outlook.com
URI: https://cybersecai.co.uk