Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: September 29, 2026

R. Sharif
CyberSecAI Ltd

March 29, 2026

ATTP: Agent Trust Transport Protocol for Secure Agent-to-Server Communication

draft-sharif-attp-agent-trust-transport-00

Abstract

This document specifies ATTP (Agent Trust Transport Protocol), a synchronous request-response protocol for communication between autonomous AI agents and web API servers. ATTP operates as an application-layer protocol over HTTP, adding mandatory cryptographic identity verification, per-message signing, trust-gated access control, and tamper-evident audit trail generation to every agent-server interaction.

ATTP defines five protocol-layer headers for requests (X-Agent-Trust, X-Agent-Signature, X-Agent-Nonce, X-Agent-Timestamp, X-ATTP-Version) and three for responses (X-Server-Signature, X-Server-Nonce, X-Server-Timestamp) that carry an Agent Passport (JWT-based identity credential), ECDSA P-256 digital signatures, cryptographic nonces, and timestamps. Server middleware verifies all cryptographic properties before application code executes.

ATTP has no insecure mode. Every request MUST carry a valid Agent Passport. Every request body MUST be signed. Every response body MUST be signed. Every interaction MUST be recorded in a hash-chained audit trail. The protocol defines a URL scheme (attp://) and is fully backward-compatible with existing HTTP infrastructure.

ATTP is the synchronous counterpart to the Agent Transport Protocol (ATP). ATP handles asynchronous store-and-forward agent delivery; ATTP handles real-time request-response API communication. Both share the same identity model, trust framework, and cryptographic primitives.

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 September 29, 2026.

Table of Contents

1. Introduction

1.1. The Problem

The Hypertext Transfer Protocol (HTTP) [RFC9110] was designed in 1991 for human users operating web browsers to request and receive hypertext documents from servers. HTTP provides no built-in mechanism for caller identity verification, no per-message signing, no trust-level evaluation, and no audit trail generation. Over three decades, security was progressively bolted on: TLS [RFC8446] for transport encryption (1995), HTTP Authentication [RFC7235] for basic credentials (1999), OAuth 2.0 [RFC6749] for delegated authorization (2012), and DPoP [RFC9449] for proof-of-possession (2023).

This layered approach created a protocol stack where security is optional, fragmented, and human-centric. HTTP has both insecure (http://) and secure (https://) variants. OAuth assumes a human who can interact with browser-based consent flows. TLS protects the transport channel but not the application-layer message -- once a request passes through a TLS-terminating proxy, CDN, or load balancer, the message is decrypted and forwarded in plaintext within the server's infrastructure.

Autonomous AI agents are now the fastest-growing consumers of web APIs. Industry projections indicate that agent-initiated API calls will exceed human-initiated API calls by 2028. These agents cannot participate in browser-based OAuth consent flows, do not have browser sessions, and require machine-to-machine identity credentials rather than human-delegated access tokens.

There exists a need for a synchronous request-response protocol purpose-built for autonomous AI agents that mandates cryptographic identity verification, signs every message at the application layer, evaluates trust at the protocol layer, generates audit trails automatically, and operates over existing HTTP infrastructure.

1.2. Design Goals

ATTP is designed with the following goals:

- Mandatory security: no insecure mode, no optional signing, no anonymous access.
- Bidirectional signing: both request and response are signed at the application layer.
- Trust-gated access: protocol-layer trust evaluation before application code executes.
- Automatic audit: every interaction recorded in a tamper-evident hash-chained trail.

- Backward compatibility: ATTP messages are valid HTTP messages that traverse existing infrastructure.
- Minimal integration: single middleware registration line in existing web frameworks.
- Federated identity: no central authority required for Agent Passport issuance or key discovery.

1.3. Relationship to ATP

ATTP is the synchronous counterpart to the Agent Transport Protocol (ATP) [draft-sharif-agent-transport-protocol]. The relationship can be summarized as:

- ATP is to ATTP as SMTP is to HTTP.
- ATP handles asynchronous, store-and-forward agent delivery (analogous to email).
- ATTP handles synchronous, real-time request-response API communication (analogous to phone calls).

Both protocols share the same identity model (Agent Passports), the same trust level framework (L0 through L4), the same cryptographic primitives (ECDSA P-256), and the same design philosophy (mandatory security, no insecure mode). They differ in delivery semantics: ATP uses a dedicated session protocol with DNS-based routing via Agent eXchange (AX) records, while ATTP runs over existing HTTP infrastructure.

An agent operating in a complex environment might use both protocols: ATTP to call web APIs for real-time data retrieval and transaction processing, and ATP to transmit itself to a different runtime for continued execution.

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 that makes independent decisions about which APIs to call and operates without direct human interaction for each request.

Agent Passport: A JWT-structured credential signed with ECDSA P-256 that binds an agent’s identity, trust level, capabilities, and public key into a single verifiable credential.

ATTP Agent: The client-side component within an autonomous agent that constructs ATTP-compliant requests, including signing the request body and attaching the Agent Passport.

ATTP Middleware: The server-side component that intercepts incoming requests, performs all ATTP verification steps (passport verification, trust evaluation, signature verification, replay protection), and either forwards verified requests to application code or rejects them.

ATTP Server: A web API server configured with ATTP middleware that enforces ATTP protocol requirements.

Trust Level: A hierarchical classification (L0 through L4) reflecting the cryptographic verification strength of an agent's identity and message integrity. Trust levels are asserted by the passport issuer and signed into the Agent Passport.

Passport Issuer: An entity that verifies agent identities, assigns trust levels, and issues signed Agent Passports. Any entity MAY operate as a passport issuer (federated model).

Nonce: A cryptographically random value (minimum 128 bits, hex-encoded) included in each request to prevent replay attacks.

JWKS: JSON Web Key Set, a document containing one or more public keys used for signature verification, as defined in [RFC7517].

Canonical JSON: JSON serialized according to JSON Canonicalization Scheme (JCS) [RFC8785], providing deterministic byte-level representation for signing.

Hash Chain: A sequence of records where each record contains a SHA-256 hash of the previous record, making the sequence tamper-evident.

3. Protocol Overview

3.1. Architecture

ATTP operates as an application-layer protocol over HTTP. ATTP messages are standard HTTP messages with additional headers and body-level signing. ATTP does not define a new transport mechanism; it leverages existing HTTP transport infrastructure (TCP connections, TLS encryption, HTTP/1.1 or HTTP/2 or HTTP/3 framing) while adding mandatory security properties at the application layer.

The architecture comprises three components:

- ATTP Agent: constructs signed requests with Agent Passport and transmits them over HTTPS.
- ATTP Middleware: server-side interceptor that verifies all cryptographic properties before forwarding to application code.
- Application Code: the API endpoint handler that processes verified requests, unaware of ATTP verification details.

Every request carries an Agent Passport and a message signature. Every response is signed by the server. The middleware enforces these requirements; application code receives only verified requests.

3.2. Mandatory Security

ATTP has no insecure mode. The following properties are non-optional protocol requirements:

- Every request MUST carry a valid Agent Passport.

- Every request body MUST be signed by the agent.
- Every response body MUST be signed by the server.
- Every interaction MUST be recorded in the audit trail.
- Trust level MUST be evaluated before application code executes.

There is no anonymous mode, no unsigned mode, and no bypass mechanism. Unlike HTTP, which has both http:// (insecure) and https:// (secure), ATTP has only attp:// which inherently implies mandatory cryptographic signing, identity verification, and trust evaluation.

3.3. URL Scheme

ATTP defines the URL scheme attp:// to distinguish agent-to-server requests from conventional HTTP requests. An ATTP URL takes the form:

```
attp://hostname[:port]/path[?query]
```

Example:

```
attp://api.example.com/v1/charges
```

The attp:// scheme signals that: (a) the request originates from an autonomous agent; (b) ATTP security requirements are mandatory; and (c) the server MUST process the request through ATTP middleware.

At the transport level, ATTP requests are transmitted over HTTPS. The attp:// scheme is resolved to an HTTPS connection by the ATTP agent component. The default port is 8443.

4. ATTP Headers

4.1. Request Headers

ATTP defines five mandatory request headers:

X-Agent-Trust: Contains the agent's Agent Passport, a JWT-structured credential signed with ECDSA P-256. The passport carries identity claims including subject, issuer, trust level, capabilities, public key, and expiration.

Format: Base64url-encoded JWT (header.payload.signature)

Example:

```
X-Agent-Trust: eyJhbGciOiJFUzI1NiIsInR5cCI6IkpX
VCIsImtpZCI6InRhLWtleS0wMDEifQ.eyJzdWIiOiJhZ2
VudC1hbHBoYS0wMDEiLCJpc3MiOiJ0cnVzdC5leGFtcG
xlLmNvbSIsInRydXN0X2xldmVsIjoiTDIiLCJleHAiOjE
3NDMzMDAwMDB9.MEUCIQDx...
```

X-Agent-Signature: Contains the Base64url-encoded ECDSA P-256 signature over the request body. For JSON bodies, the signature is computed over the JCS-canonicalized [RFC8785] form. For non-JSON bodies, the signature is computed over the raw bytes. For bodiless requests (GET, HEAD), the signature covers a canonical string comprising method, path, nonce, and timestamp.

Encoding: IEEE P1363 fixed-length r||s format per

[RFC7518] Section 3.4, with low-S normalization.

Format: Base64url string

X-Agent-Nonce: Contains a cryptographically random nonce, hex-encoded, with a minimum of 128 bits of entropy (32 hex characters). The nonce MUST be unique per request.

Format: Hex-encoded string, minimum 32 characters

Example:
X-Agent-Nonce: a1b2c3d4e5f6a7b8c9d0e1f2a3b4c5d6

X-Agent-Timestamp: Contains an ISO 8601 timestamp [RFC3339] recording the moment the agent signed the request.

Format: ISO 8601 / RFC 3339 timestamp

Example:
X-Agent-Timestamp: 2026-03-29T14:30:00.000Z

X-ATTP-Version: Contains the ATTP protocol version string.

Format: Major.Minor version string

Example:
X-ATTP-Version: 1.0

4.2. Response Headers

ATTP defines three mandatory response headers:

X-Server-Signature: Contains the Base64url-encoded ECDSA P-256 signature over the response body, computed by the server using its private key. The agent verifies this signature using the server’s public key obtained from the JWKS endpoint.

X-Server-Nonce: Contains a cryptographically random nonce generated by the server, hex-encoded, minimum 128 bits.

X-Server-Timestamp: Contains an ISO 8601 timestamp recording the moment the server signed the response.

5. Request Lifecycle

The ATTP request lifecycle comprises eleven steps. Each step is mandatory and MUST NOT be skipped or deferred.

Step 1: Request Construction

The ATTP agent constructs the request payload (HTTP request body) containing the data to be sent. The payload MAY be JSON, form data, binary, or any content type supported by HTTP.

Step 2: Request Body Signing

The ATTP agent computes a digital signature over the request body using its ECDSA P-256 private key. For JSON payloads, the canonical form is computed using JCS [RFC8785]: keys sorted lexicographically, no extraneous whitespace, UTF-8 encoding. For non-JSON payloads, the raw byte sequence is signed. The signature uses IEEE P1363 fixed-length r||s encoding with low-S

normalization. The resulting signature is Base64url-encoded.

For bodiless requests (GET, HEAD, DELETE without body), the agent signs a canonical string:

```
METHOD + "\n" + PATH + "\n" + NONCE + "\n" + TIMESTAMP
```

Step 3: Agent Passport Attachment

The ATTP agent attaches its Agent Passport in the X-Agent-Trust header. The passport is a JWT containing the agent's identity claims, signed by the passport issuer.

Step 4: Nonce and Timestamp Generation

The ATTP agent generates a cryptographically random nonce (minimum 128 bits, hex-encoded) and an ISO 8601 timestamp. The nonce is placed in X-Agent-Nonce; the timestamp in X-Agent-Timestamp. Both values are included in the signed content (Step 2) to bind the signature to a specific moment and prevent replay.

Step 5: Request Transmission

The ATTP agent transmits the HTTP request to the server's URL with all ATTP headers (X-Agent-Trust, X-Agent-Signature, X-Agent-Nonce, X-Agent-Timestamp, X-ATTP-Version) alongside standard HTTP headers. The request is transmitted over HTTPS.

Step 6: Server Middleware Interception

The ATTP middleware intercepts the incoming request before it reaches the application's route handler. The middleware checks for X-ATTP-Version to determine if the request is an ATTP request. If absent, behavior is configurable:

- Strict mode: reject (ATTP-only server).
- Permissive mode: pass through as regular HTTP.
- Upgrade mode: respond with protocol upgrade suggestion.

Step 7: Agent Passport Verification

The middleware extracts the Agent Passport from X-Agent-Trust, decodes the JWT, and verifies its ECDSA P-256 signature against the issuer's public key. The middleware verifies:

- The passport has not expired.
- The passport's issuer is in the server's trusted list.
- The passport's claims are internally consistent.

The issuer's public key is resolved via the issuer's JWKS endpoint or a locally cached copy.

Step 8: Trust Level Evaluation

The middleware extracts the agent's trust level from the verified passport and compares it against the minimum

trust level required by the target endpoint. If the agent's trust level is below the minimum, the middleware rejects the request with HTTP 403 and a structured error response (see Section 6.3).

Step 9: Request Signature Verification

The middleware extracts the signature from X-Agent-Signature and verifies it against the request body using the agent's public key from the passport. The middleware also verifies:

- The nonce has not been previously used.
- The timestamp is within the acceptable window.

If verification fails, the request is rejected.

Step 10: Response Signing

After application code generates a response, the middleware intercepts the outgoing response. The middleware computes an ECDSA P-256 signature over the response body using the server's private key, generates a server nonce and timestamp, and attaches: X-Server-Signature, X-Server-Nonce, X-Server-Timestamp.

Step 11: Audit Trail Recording

The middleware records the complete request-response pair in the tamper-evident audit trail. The audit record includes the agent's identity, trust level, request and response hashes, signatures, and a hash pointer to the previous record.

6. Trust-Gated Access Control

ATTP enforces trust-level-based access control at the protocol layer, before application code executes. This is fundamentally different from application-layer authorization (such as OAuth scope checking): ATTP trust evaluation occurs in protocol middleware and rejects untrusted requests before they consume application resources.

Trust levels reflect the cryptographic verification strength of the agent's identity, not the permissions granted. Trust answers "how confident is the server in the agent's identity and message integrity?" while permissions answer "what is the agent allowed to do?" ATTP addresses the former; the application layer addresses the latter.

6.1. Trust Level Definitions

ATTP defines five hierarchically ordered trust levels, derived from the MCPS trust level framework [draft-sharif-mcps-secure-mcp]:

L0 (No Verification): The agent has presented a passport but the issuer cannot be verified, or the agent has no established history. L0 is the floor; ATTP does not permit passportless requests.

L1 (Identity Verified): The agent's passport is signed by a trusted issuer that has verified the agent's identity.

L2 (Signed Messages): The agent's passport is verified and

the request is signed with a valid ECDSA P-256 signature. L2 is the baseline for most API operations.

L3 (Full Verification with Revocation Checking): Passport verified, request signed, and revocation checking performed against the issuer's revocation endpoint. Appropriate for sensitive operations such as financial transactions.

L4 (Mutual Authentication with Hardware-Backed Keys): All L3 requirements plus attestation that the agent's private key resides in a hardware security module (HSM) or secure enclave. Appropriate for critical infrastructure operations.

6.2. Per-Endpoint Trust Declaration

Each API endpoint declares its minimum trust level as part of its configuration, not its application code. The ATTP middleware reads the declared minimum and compares it against the agent's trust level (where L0=0, L1=1, L2=2, L3=3, L4=4).

Examples:

- Public product catalogue: L1 minimum
- Standard API operations: L2 minimum
- Payment processing: L3 minimum
- Critical infrastructure: L4 minimum

6.3. Rejection Semantics

When the agent's trust level is below the endpoint's minimum, the middleware MUST respond with HTTP 403 Forbidden and the following JSON body:

```
{
  "error": "insufficient_trust_level",
  "required_level": "L3",
  "agent_level": "L1",
  "message": "Agent trust level insufficient"
}
```

This rejection occurs at the protocol layer. Application code is never invoked for insufficient-trust requests.

7. Per-Message Signing

ATTP mandates per-message signing for both requests and responses, providing bidirectional application-layer message integrity. This is a distinguishing property: HTTP does not sign messages, TLS signs only the transport channel, and most API authentication schemes sign only the request. ATTP signs both directions.

7.1. Request Signing

The agent computes an ECDSA P-256 signature over the canonicalized request body, the nonce, and the timestamp. The signature binds the message content to the agent's identity (via its private key) and to a specific point in time (via nonce and timestamp).

If any byte of the request body is modified after signing -- by a proxy, CDN, API gateway, or any other intermediary -- the signature verification at the server will fail.

The signing input for JSON request bodies:

```
JCS(request_body) + "\n" + nonce + "\n" + timestamp
```

The signing input for non-JSON request bodies:

```
raw_bytes(request_body) + "\n" + nonce + "\n" + timestamp
```

All signatures use ECDSA P-256 with IEEE P1363 r||s encoding and low-S normalization per [RFC7518] Section 3.4.

7.2. Response Signing

The server computes an ECDSA P-256 signature over the response body using the server's private key. The agent verifies this signature using the server's public key obtained from the JWKS endpoint (Section 9).

Bidirectional signing provides end-to-end integrity from agent to server and from server to agent, regardless of intermediaries in the network path.

7.3. Canonical JSON

For JSON payloads, ATTP uses JSON Canonicalization Scheme (JCS) as specified in [RFC8785] to produce deterministic byte-level representations for signing:

- Object keys sorted lexicographically by Unicode code point.
- No extraneous whitespace.
- UTF-8 encoding.
- Number serialization per ECMAScript rules.

JCS ensures that semantically identical JSON documents produce identical byte sequences, preventing signature verification failures caused by formatting differences.

7.4. Signature Verification

Verification MUST fail-closed: any verification error MUST result in request rejection. The middleware MUST NOT fall back to unsigned processing.

ECDSA P-256 was selected for its combination of 128-bit equivalent security strength, performance, compact key size (suitable for HTTP headers), and ubiquity (supported by all major cryptographic libraries, HSMs, and cloud KMS services).

8. Agent Passport

The Agent Passport is the cornerstone of ATTP identity verification. Every ATTP request MUST carry a valid Agent Passport. There is no anonymous mode and no bypass.

8.1. Passport Structure

The passport is a JWT [RFC7519] comprising three sections:

Header: Specifies the signing algorithm (ES256 for ECDSA P-256) and the key identifier (kid) of the issuer's signing key.

Payload: Contains identity claims (see Section 8.2).

Signature: Computed by the issuer over the header and payload using the issuer's ECDSA P-256 private key.

8.2. Mandatory Fields

The Agent Passport payload MUST contain:

sub (Subject): The agent's unique identifier.
Example: "agent-alpha-001"

iss (Issuer): The identity provider that issued the passport.
Example: "trust.example.com"

iat (Issued At): Unix timestamp of passport issuance.

exp (Expiration): Unix timestamp of passport expiration.

trust_level: One of "L0", "L1", "L2", "L3", "L4".

capabilities: Array of capability strings the agent possesses.
Example: ["read", "write", "payment"]

The passport payload SHOULD contain:

owner: The individual or organization that owns the agent.

agent_type: Classification: "autonomous", "semi-autonomous", or "supervised".

pub_key: The agent's ECDSA P-256 public key in JWK format [RFC7517] or a URI reference to the key.

origin: The domain from which the agent operates, for origin binding.

8.3. Verification Procedure

The ATTP middleware MUST verify the Agent Passport through the following steps:

1. Decode the JWT and extract header, payload, and signature.

2. Extract the issuer (iss) claim and resolve the issuer's public key via the issuer's JWKS endpoint or a locally cached copy.

3. Verify the JWT signature against the header and payload using the issuer's public key.

4. Verify that the current time is between iat and exp.

5. Verify that the issuer is in the server's list of trusted issuers.

6. Optionally perform origin binding verification: confirm that the agent's declared origin matches the network

origin of the request.

7. For L3 and L4 trust levels, perform revocation checking by querying the issuer's revocation endpoint.

If any step fails, the middleware MUST reject the request with HTTP 401 and an appropriate error code.

8.4. Relationship to MCPS Agent Passports

ATTP uses the Agent Passport format defined in MCPS [draft-sharif-mcps-secure-mcp] Section 4, extended with HTTP-specific claims for transport context. MCPS defines how to issue, sign, and verify passports; ATTP defines when and where these operations occur within the HTTP request-response lifecycle.

9. Key Discovery

9.1. JWKS Endpoint

ATTP servers MUST publish their public keys at:

/.well-known/agent-trust-keys

This endpoint serves a JSON Web Key Set (JWKS) document [RFC7517] containing one or more ECDSA P-256 public keys used for response signing.

The JWKS document contains JWK objects with:

- kty: "EC" (Elliptic Curve)
- crv: "P-256"
- x, y: public key coordinates, Base64url-encoded
- kid: key identifier for key selection
- use: "sig" (signing)
- alg: "ES256" (ECDSA P-256)

Example JWKS document:

```
{
  "keys": [{
    "kty": "EC",
    "crv": "P-256",
    "kid": "server-key-001",
    "use": "sig",
    "alg": "ES256",
    "x": "f83OJ3D2xF1Bg8vub9tLe1gHMzV76e8Tus9uPHvRVEU",
    "y": "x_FEzRu9m36HLN_tue659LNpXW6pCyStikYjKIWI5a0"
  }]
}
```

The well-known path follows [RFC8615] conventions. The path "agent-trust-keys" avoids collision with other well-known endpoints such as /.well-known/jwks.json.

9.2. Key Caching

Agents SHOULD cache the JWKS document to avoid fetching it on every request. The HTTP Cache-Control header on the JWKS response indicates the cache duration. A typical

cache duration is one hour.

Agents MUST refresh the cached JWKS document if they encounter a response signed with a key identifier (kid) not present in their cached document.

9.3. Key Rotation

Key rotation is supported through the JWKS endpoint. When a server rotates its signing key:

1. Publish both old and new keys in the JWKS document.

2. Begin signing responses with the new key.

3. After a transition period, remove the old key.

This allows agents with cached old keys to continue verifying responses during the transition.

10. Replay Protection

10.1. Nonce Requirements

Each ATTP request MUST carry a unique nonce in the X-Agent-Nonce header. The nonce MUST be cryptographically random with a minimum of 128 bits of entropy, hex-encoded to a 32-character string. The nonce is included in the signed content, binding it cryptographically to the request.

An attacker who captures a valid ATTP request cannot modify the nonce without invalidating the signature.

10.2. Timestamp Window

The X-Agent-Timestamp header MUST contain an ISO 8601 timestamp [RFC3339]. The server MUST reject requests whose timestamp is outside the acceptable window.

The default window is plus or minus 300 seconds (5 minutes) from the server’s current time. Servers MAY configure a different window but MUST NOT exceed plus or minus 600 seconds.

The timestamp window bounds replay vulnerability: even if an attacker captures a request, they have at most the window duration to attempt replay, and the nonce cache prevents even that.

10.3. Nonce Store Requirements

The server MUST maintain a nonce store that records all nonces received within the timestamp validity window.

- On receipt: check if nonce exists in store. If yes, reject as replay (HTTP 409). If no, add to store.

- Nonce entries MUST be expired after the timestamp window closes, bounding memory requirements.

- For multi-instance deployments, the nonce store MUST be shared across instances (e.g., Redis).

The combination of nonce uniqueness and timestamp window provides robust replay protection with bounded resource consumption.

11. Audit Trail

11.1. Automatic Recording

The ATTP middleware MUST record every request-response pair in a tamper-evident audit trail. This recording is automatic; application code MUST NOT need to implement logging for ATTP interactions.

Each audit record contains:

- Record identifier (UUID v4) [RFC9562]
- Agent identifier (from verified passport)
- Agent trust level
- Agent owner (from passport)
- Request timestamp
- HTTP method and request path
- SHA-256 hash of request body
- Request signature (from X-Agent-Signature)
- Response status code
- SHA-256 hash of response body
- Response signature (from X-Server-Signature)
- Processing duration (milliseconds)
- Hash pointer to previous audit record

The body itself is not stored (to avoid retaining sensitive payloads); the SHA-256 hash enables later verification.

11.2. Hash-Chained Records

Each audit record contains a SHA-256 hash of the previous record's serialized content, creating a hash chain per [draft-sharif-agent-audit-trail]. The first record has a null hash pointer.

This hash chain is tamper-evident: if any record is modified, deleted, or inserted, the chain breaks and tampering is detectable by any party that verifies it.

11.3. Export Formats

ATTP audit trails MUST support export in the following formats:

JSONL: One JSON object per line, suitable for log aggregation systems and streaming processors.

Syslog: RFC 5424 structured data format, suitable for enterprise SIEM integration.

12. URL Scheme

12.1. Scheme Definition

ATTP defines the URI scheme "attp" for identifying resources accessible via the Agent Trust Transport Protocol.

The scheme is registered per [RFC7595].

Scheme syntax:

```
attp-URI = "attp://" authority path-abempty
           [ "?" query ] [ "#" fragment ]
```

The "attp" scheme implies:

- Mandatory Agent Passport verification
- Mandatory per-message signing (both directions)
- Mandatory trust-level evaluation
- Mandatory audit trail recording
- Transport over HTTPS (TLS required)

There is no insecure counterpart. Unlike HTTP (http:// vs https://), the attp:// scheme inherently implies full security.

12.2. Default Port

The default port for ATTP is 8443. This port is selected to avoid conflict with the standard HTTPS port (443) while remaining within the common range for secure services.

When the port is omitted from the URI, the ATTP agent component MUST connect to port 8443.

12.3. URI Format

```
attp://host[:port]/path[?query]
```

Examples:

```
attp://api.example.com/v1/charges
attp://api.example.com:9443/v1/users?limit=10
attp://trust.internal.corp/agents/register
```

13. Backward Compatibility

13.1. HTTP Infrastructure

ATTP messages are standard HTTP messages. All ATTP-specific information is carried in HTTP headers and in the cryptographic relationship between headers and body. No modifications to HTTP framing, body encoding, or transport semantics are required.

Existing infrastructure forwards ATTP transparently:

- Load balancers (HAProxy, NGINX, AWS ALB) forward all headers including X-Agent-* headers.
- CDNs (Cloudflare, Fastly, Akamai) cache and forward ATTP responses, preserving X-Server-* headers.
- Reverse proxies and API gateways forward ATTP traffic.

- WAFs inspect ATTP as standard HTTP traffic with additional headers available for custom rules.

- Monitoring tools capture ATTP as standard HTTP with ATTP headers available for agent-specific analysis.

ATTP's per-message signing provides security that survives transit through TLS-terminating intermediaries. When a request passes through a TLS-terminating load balancer, the ATTP signature remains verifiable at the backend server, confirming the body was not modified in transit.

13.2. Upgrade Path

Servers transitioning from HTTPS to ATTP can operate in three modes:

Strict: Reject all non-ATTP requests (HTTP 426 Upgrade Required).

Permissive: Accept both ATTP and standard HTTP requests, applying ATTP verification only when ATTP headers are present.

Upgrade: Accept standard HTTP requests but include an Upgrade header in responses suggesting ATTP.

Non-ATTP servers that receive requests with X-Agent-* headers will ignore the unknown headers and process the request as standard HTTP. This enables gradual adoption.

14. Error Handling

ATTP defines specific error responses for protocol-level failures, using standard HTTP status codes with structured JSON error bodies. All error responses are machine-readable for autonomous agent consumption.

400 Bad Request - missing_attp_headers:

```
{
  "error": "missing_attp_headers",
  "missing_headers": ["X-Agent-Trust",
                      "X-Agent-Signature"]
}
```

Remedial action: agent adds missing headers.

401 Unauthorized - invalid_passport:

```
{
  "error": "invalid_passport",
  "reason": "expired"
}
```

Reasons: "signature_invalid", "expired", "issuer_untrusted", "malformed".
Remedial action: obtain new passport from issuer.

401 Unauthorized - invalid_signature:

```
{
  "error": "invalid_signature",
  "reason": "signature_mismatch"
}
```

Reasons: "signature_mismatch", "key_mismatch", "canonicalization_error".

403 Forbidden - insufficient_trust_level:

```
{
  "error": "insufficient_trust_level",
  "required_level": "L3",
  "agent_level": "L1"
}
```

Remedial action: request higher trust level or find alternative endpoint.

408 Request Timeout - timestamp_expired:

```
{
  "error": "timestamp_expired"
}
```

Indicates clock skew or delayed request.

409 Conflict - nonce_reuse:

```
{
  "error": "nonce_reuse"
}
```

Indicates potential replay attack. No additional detail to avoid information leakage.

426 Upgrade Required - attp_required:

```
{
  "error": "attp_required",
  "upgrade": "ATTP/1.0",
  "documentation": "https://example.com/attp"
}
```

Server operates in strict mode; ATTP headers required.

15. Security Considerations

Mandatory security model: ATTP eliminates the "forgot to enable TLS" class of vulnerabilities. There is no insecure mode, no optional signing, and no configuration that disables security. Every protocol interaction requires identity, signing, and audit.

Trust evaluation at protocol layer: Trust-gated access prevents application-layer bypasses. An application vulnerability cannot bypass trust evaluation because trust is enforced in protocol middleware before application code executes.

Bidirectional signing: Response signing prevents response tampering by intermediaries. Unlike TLS, which protects only the transport channel, ATTP signatures survive transit through TLS-terminating proxies, CDNs, and load balancers.

Replay protection: The combination of unique nonce and bounded timestamp window prevents replay attacks with bounded resource consumption for the nonce store.

Privacy: Agent identity is visible to the server by design.

This is intentional for audit and accountability purposes. Servers MUST protect agent identity data per applicable privacy regulations. Agents operating in privacy-sensitive contexts SHOULD use pseudonymous identifiers in their passports.

Clock skew: The timestamp window (default 300 seconds) accommodates reasonable clock drift. Agents and servers SHOULD synchronize clocks via NTP [RFC5905].

Key compromise: If an agent's private key is compromised, the passport issuer SHOULD revoke the agent's passport. Servers performing L3+ verification will detect the revocation. Key rotation procedures are specified in Section 9.3.

Algorithm agility: The X-ATTP-Version header enables future protocol versions to introduce post-quantum signature algorithms without breaking backward compatibility with existing deployments.

Nonce store exhaustion: Attackers may attempt to exhaust the nonce store by sending large volumes of unique nonces. The timestamp window bounds the store's memory requirements. Servers SHOULD implement rate limiting on ATTP requests.

16. IANA Considerations

16.1. ATTP URI Scheme Registration

This document requests registration of the "attp" URI scheme per [RFC7595]:

Scheme name: attp

Status: Permanent

Applications/protocols that use this scheme: Agent Trust Transport Protocol (ATTP)

Contact: R. Sharif (contact@agentsign.dev)

Change Controller: IESG

References: This document

16.2. Port Number Registration

This document requests registration of a TCP port number for ATTP:

Service Name: attp

Transport Protocol: TCP

Assignee: R. Sharif (contact@agentsign.dev)

Contact: R. Sharif (contact@agentsign.dev)

Description: Agent Trust Transport Protocol

Reference: This document

Port Number: 8443

16.3. ATTP Header Field Registration

This document requests registration of the following HTTP header fields in the "Hypertext Transfer Protocol (HTTP) Field Name Registry" maintained by IANA:

Header Field Name	Status	Reference
X-Agent-Trust	standard	this doc
X-Agent-Signature	standard	this doc
X-Agent-Nonce	standard	this doc
X-Agent-Timestamp	standard	this doc
X-ATTP-Version	standard	this doc
X-Server-Signature	standard	this doc
X-Server-Nonce	standard	this doc
X-Server-Timestamp	standard	this doc

17. References

17.1. 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, <https://www.rfc-editor.org/info/rfc2119>.

[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>.

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

[RFC7519] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", RFC 7519, DOI 10.17487/RFC7519, May 2015, <https://www.rfc-editor.org/info/rfc7519>.

[RFC7595] Thaler, D., Hansen, T., and T. Hardie, "Guidelines and Registration Procedures for URI Schemes", BCP 35, RFC 7595, DOI 10.17487/RFC7595, June 2015, <https://www.rfc-editor.org/info/rfc7595>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

[RFC8615] Nottingham, M., "Well-Known Uniform Resource Identifiers (URIs)", RFC 8615, DOI 10.17487/RFC8615, May 2019, <https://www.rfc-editor.org/info/rfc8615>.

[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>.

[RFC9110] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Semantics", STD 97, RFC 9110, DOI 10.17487/RFC9110, June 2022, <https://www.rfc-editor.org/info/rfc9110>.

[RFC9562] Davis, K., Peabody, B., and P. Leach, "Universally Unique IDentifiers (UUIDs)", RFC 9562, DOI 10.17487/RFC9562, May 2024, <https://www.rfc-editor.org/info/rfc9562>.

17.2. Informative References

[RFC5905] Mills, D., Martin, J., Ed., Burbank, J., and W. Kasch, "Network Time Protocol Version 4: Protocol and Algorithms Specification", RFC 5905, DOI 10.17487/RFC5905, June 2010, <https://www.rfc-editor.org/info/rfc5905>.

[RFC6749] Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012, <https://www.rfc-editor.org/info/rfc6749>.

[RFC7235] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Authentication", RFC 7235, DOI 10.17487/RFC7235, June 2014, <https://www.rfc-editor.org/info/rfc7235>.

[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>.

[draft-sharif-mcps-secure-mcp] Sharif, R., "MCPS: Cryptographic Security Layer for the Model Context Protocol", Internet-Draft draft-sharif-mcps-secure-mcp-02, March 2026, <https://datatracker.ietf.org/doc/draft-sharif-mcps-secure-mcp/>.

[draft-sharif-agent-payment-trust] Sharif, R., "Agent Payment Trust: Cryptographic Payment Authorization for Autonomous AI Agents", Internet-Draft draft-sharif-agent-payment-trust-00, March 2026, <https://datatracker.ietf.org/doc/draft-sharif-agent-payment-trust/>.

[draft-sharif-agent-audit-trail] Sharif, R., "Agent Audit Trail: Hash-Chained Interaction Records for AI Agent Systems", Internet-Draft draft-sharif-agent-audit-trail-00, March 2026,

            <https://datatracker.ietf.org/doc/
            draft-sharif-agent-audit-trail/>.

[draft-sharif-agent-transport-protocol]
            Sharif, R., "ATP: Agent Transport Protocol for
            Asynchronous Agent Delivery",
            Internet-Draft
            draft-sharif-agent-transport-protocol-00,
            March 2026,
            <https://datatracker.ietf.org/doc/
            draft-sharif-agent-transport-protocol/>.

Appendix A. Example ATTP Exchange

The following is a complete ATTP request-response exchange for a payment processing endpoint.

A.1. Request

```
POST /v1/charges HTTP/1.1
Host: api.example.com
Content-Type: application/json
X-ATTP-Version: 1.0
X-Agent-Trust: eyJhbGciOiJFUzI1NiIsInR5cCI6IkpXVCIs
  ImtpZCI6InRhLWtleS0wMDEifQ.eyJzdWIiOiJwYXltZW50
  LWJvdC0wMDEiLCJpc3MiOiJ0cnVzdC5leGFtcGxlLmNvbSI
  sImlhdCI6MTcxMTcwMDAwMCwiZXhwIjoxNzExNzg2NDAwLCJ
  0cnVzdF9sZXZlbCI6IkwzIiwiY2FwYWJpbGl0aWVzIjpbIn
  JlYWQiLCJ3cml0ZSIsInBheW1lbnQiXSwib3duZXIiOiJBY2
  1lIENvcnAiLCJhZ2VudF90eXBlIjoiYXV0b25vbW91cyJ9.
  MEUCIQDxR2s...signature...
X-Agent-Signature: MEUCIQCvK7n...signature-over-body
X-Agent-Nonce: a1b2c3d4e5f6a7b8c9d0e1f2a3b4c5d6
X-Agent-Timestamp: 2026-03-29T14:30:00.000Z

{"amount":5000,"currency":"usd","description":"Widget"}
```

A.2. Decoded Agent Passport Payload

```
{
  "sub": "payment-bot-001",
  "iss": "trust.example.com",
  "iat": 1711700000,
  "exp": 1711786400,
  "trust_level": "L3",
  "capabilities": ["read", "write", "payment"],
  "owner": "Acme Corp",
  "agent_type": "autonomous",
  "pub_key": {
    "kty": "EC",
    "crv": "P-256",
    "x": "f83OJ3D2xF1Bg8vub9tLe...",
    "y": "x_FEzRu9m36HLN_tue659L..."
  },
  "origin": "agents.acme.com"
}
```

A.3. Response

```
HTTP/1.1 200 OK
Content-Type: application/json
X-Server-Signature: MEQCIFhN...server-signature...
X-Server-Nonce: f1e2d3c4b5a6f7e8d9c0b1a2f3e4d5c6
X-Server-Timestamp: 2026-03-29T14:30:00.150Z

{"id":"ch_abc123","amount":5000,"currency":"usd",
```

```
    "status":"succeeded"}
```

A.4. Middleware Verification Sequence

1. Extract X-ATTP-Version: "1.0" - PASS
2. Decode X-Agent-Trust JWT
3. Fetch issuer JWKS: trust.example.com /.well-known/agent-trust-keys
4. Verify passport signature - PASS
5. Check passport expiration - PASS (not expired)
6. Check issuer trust - PASS (trusted)
7. Extract trust_level: L3
8. Endpoint /v1/charges requires: L3
9. Trust check: L3 >= L3 - PASS
10. Verify X-Agent-Signature against body - PASS
11. Check X-Agent-Nonce not in store - PASS (unique)
12. Check X-Agent-Timestamp within window - PASS
13. Forward to application handler
14. Sign response body with server key
15. Record audit trail entry (hash-chained)

Appendix B. Comparison with HTTP/OAuth

The following table maps HTTP and OAuth features to their ATTP equivalents.

HTTP/OAuth	ATTP Equivalent
URL Scheme http:// (insecure) https:// (secure)	(no equivalent, no insecure mode) attp:// (mandatory security)
Identity OAuth Bearer token (delegated human auth)	Agent Passport (JWT with trust level, signed by issuer, mandatory)
Request Integrity None (HTTP) TLS (transport only)	X-Agent-Signature (ECDSA P-256 over body, survives TLS termination)
Response Integrity None (HTTP) TLS (transport only)	X-Server-Signature (ECDSA P-256 over response)
Access Control OAuth scopes (application layer)	Trust levels (L0-L4) at protocol layer before app code executes
Replay Protection None (HTTP) CSRF tokens (web only)	X-Agent-Nonce + timestamp (cryptographically bound to signature)
Audit Application-layer logs (developer-implemented)	Automatic hash-chained audit trail (protocol layer, no dev effort)
Key Discovery OAuth .well-known/openid	/.well-known/agent-trust

```
|  -configuration           |  -keys (JWKS, RFC 7517)   |
+---------------------------+---------------------------+
| Signing Algorithm         |                           |
|  Varies (RS256, ES256,    | ECDSA P-256 only (ES256,  |
|  HS256, etc.)             |  IEEE P1363, low-S norm)  |
+---------------------------+---------------------------+
| Security Model            |                           |
|  Optional (http vs https) | Mandatory (no insecure    |
|  Configurable (OAuth)     |  mode, no bypass)         |
+---------------------------+---------------------------+
```

Appendix C. Server Middleware Implementation Guide

This appendix provides guidance for implementing ATTP middleware in common web frameworks.

C.1. Express.js (Node.js)

```
const attp = require('attp-middleware');

// Global: all endpoints require L2 minimum
app.use(attp({
  privateKey: process.env.ATTP_PRIVATE_KEY,
  keyId: 'server-key-001',
  trustedIssuers: ['trust.example.com'],
  minTrust: 'L2',
  nonceStore: 'redis://localhost:6379'
}));

// Per-endpoint: payment requires L3
app.post('/v1/charges',
  attp({ minTrust: 'L3' }),
  async (req, res) => {
    // req.agent.id = "payment-bot-001"
    // req.agent.trustLevel = "L3"
    // req.agent.owner = "Acme Corp"
    const charge = await processPayment(req.body);
    res.json(charge);
    // Response auto-signed by middleware
    // Audit record auto-generated
  }
);
```

C.2. FastAPI (Python)

```
from attp import ATTPMiddleware

app = FastAPI()
app.add_middleware(ATTPMiddleware,
    private_key=os.environ["ATTP_PRIVATE_KEY"],
    key_id="server-key-001",
    trusted_issuers=["trust.example.com"],
    min_trust="L2",
    nonce_store="redis://localhost:6379"
)

@app.post("/v1/charges")
async def create_charge(request: Request):
    agent = request.state.agent
    # agent.id, agent.trust_level, agent.owner
    return {"id": "ch_abc123", "status": "succeeded"}
```

C.3. JWKS Endpoint Setup

The middleware MUST automatically serve the JWKS endpoint at /.well-known/agent-trust-keys. The

endpoint returns:

```
HTTP/1.1 200 OK
Content-Type: application/json
Cache-Control: public, max-age=3600

{
  "keys": [{
    "kty": "EC",
    "crv": "P-256",
    "kid": "server-key-001",
    "use": "sig",
    "alg": "ES256",
    "x": "<Base64url x-coordinate>",
    "y": "<Base64url y-coordinate>"
  }]
}
```

C.4. Operating Modes

The middleware supports three operating modes:

Strict: Reject non-ATTP requests with HTTP 426. Use for ATTP-only servers.

Permissive: Accept both ATTP and regular HTTP. Apply verification only when ATTP headers present. Use during migration.

Upgrade: Accept all requests but suggest ATTP in responses via Upgrade header. Use for gradual adoption.

C.5. Nonce Store Configuration

Single instance: In-memory store (default).

Multi-instance: Redis or equivalent distributed cache. Required when running behind a load balancer.

Interface:

```
has(nonce: string): boolean
add(nonce: string, ttl: number): void
```

Author's Address

Raza Sharif
CyberSecAI Ltd
205 Regent Street
London W1B 4HB
United Kingdom

Email: contact@agentsign.dev