Network Working Group
Internet-Draft
Intended status: Informational
Expires: 28 May 2026

S. Dhir
Independent

24 November 2025

HTTP Agent Profile (HAP): Authenticated and Monetized Agent Traffic on the Web
draft-dhir-http-agent-profile-00

Abstract

Autonomous agents such as LLM-powered crawlers, browser-integrated assistants, and task-oriented bots are rapidly becoming first-class HTTP clients on the Web. Today’ s infrastructure largely assumes a human behind a browser and monetizes content through advertising and coarse subscriptions. Automated agents consume content at scale without rendering pages or viewing ads, exacerbating bot-mitigation arms races and economic misalignment between content providers and AI systems.

This document describes an HTTP Agent Profile (HAP) that enables: (1) cryptographic authentication of agent traffic using HTTP Message Signatures; (2) clear separation between human and agent traffic using privacy-preserving human tokens; and (3) protocol-level value exchange for agents via HTTP status code 402 ("Payment Required") and pluggable micropayment mechanisms. The profile reuses existing HTTP features and is designed for incremental deployment via reverse proxies, CDNs, and agent libraries.

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 May 2026.

Table of Contents

1. Introduction

Web traffic is undergoing a shift from primarily human-driven browsing to increasing volumes of autonomous agent activity. Modern agents include search crawlers, LLM-based assistants that browse on behalf of users, and specialized bots that fetch and process large amounts of content. These agents often access resources without rendering pages or viewing ads, and without participating in the economic arrangements that publishers rely on for human visitors.

Existing mechanisms for distinguishing human traffic from automated traffic rely on fragile signals such as User-Agent strings, IP address ranges, and CAPTCHAs. Sophisticated agents can mimic human behavior, use residential proxies, and even outsource CAPTCHAs to humans, making traditional bot detection increasingly ineffective.

At the same time, publishers and API providers respond with more aggressive blocking and rate limiting, which frequently catches legitimate agent services in the crossfire. The result is an adversarial "dark forest" dynamic in which neither side has clear, protocol-level tools to cooperate.

In parallel, regulators are starting to require clearer transparency about AI systems: for example, rules that users must be informed when interacting with an AI system rather than a human, and that bots must not misrepresent themselves for certain purposes. These pressures suggest that HTTP should grow more explicit mechanisms for agent identification and handling.

This document proposes an HTTP Agent Profile (HAP) as an HTTP-based approach to: (1) authenticate agent traffic using HTTP Message Signatures ([RFC9421]); (2) keep a separate "human lane" identified by privacy-preserving human tokens; and (3) enable protocol-level payments for agent access using HTTP 402. Rather than defining a new application protocol, HAP stays within HTTP semantics ([RFC9110]) and is intended to be deployed via existing HTTP infrastructure, especially reverse proxies, CDNs, and agent SDKs.

2. Conventions and 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.

The following terms are used throughout this document:

* _Agent_: An autonomous HTTP client that issues requests without a human directly driving each interaction (for example, a crawler, an LLM-based assistant, or an automated script).

* _Human client_: A human user interacting with a browser or similar user agent.

* _HTTP Agent Profile (HAP)_: A set of HTTP conventions for authenticating agents, separating human versus agent traffic, and expressing value exchange requirements for agents using HTTP 402.

* _Signed agent request_: An HTTP request that carries an HTTP Message Signature bound to an agent identifier.

* _Payment challenge_: An HTTP 402 response that encodes a machine-readable requirement for payment or other economic work before access is granted.

3. Requirements and Problem Statement

Existing approaches to managing automated traffic have several limitations: User-Agent strings are self-declared and trivial to spoof; IP-based blocking is undermined by residential proxy networks and shared hosting; and CAPTCHA-based human verification is increasingly defeated by automation or outsourced labor, while degrading user experience for real humans. Many sites treat any suspicious traffic as hostile, which often includes legitimate agents that do not identify themselves.

We identify the following requirements for a pragmatic solution:

1. _R1 Agent Identification:_ Servers should be able to cryptographically verify the origin of a request (which agent/software operator is responsible), rather than relying on self-asserted headers.

2. _R2 Human vs. Agent Separation:_ The mechanism should enable servers to treat human and agent traffic differently, without imposing undue friction (like CAPTCHAs) on legitimate human users.

3. _R3 Value Exchange:_ Servers should be able to require payment or other forms of compensation from agents at the HTTP protocol level (for example, using HTTP 402 challenges) in a machine-readable, automatable way.

4. _R4 Incremental Deployability:_ The solution must work within existing HTTP/HTTPS infrastructure (standard ports, TLS, proxies, CDNs) and allow gradual adoption with graceful fallback for non-participating clients.

5. _R5 Privacy and Openness:_ The design should accommodate both persistent agent identities (for building reputation) and ephemeral ones (for privacy), and it should not mandate that all content be behind a paywall for agents free access options should remain possible.

6. _R6 Extensibility:_ Higher-level frameworks (like robots.txt/llms.txt rules, AI metadata, or reputation systems) should be able to build on top of the mechanism, using it as a foundation for more sophisticated policy or trust layers.

In summary, the Web currently lacks a standardized method for agents to authenticate themselves or negotiate economic terms, and this proposal aims to fill that gap while meeting the above requirements.

4. Design Space: New Protocol vs HTTP Profile

One possible approach to agent traffic is to define a new application-layer protocol specifically for agents (an "Agent-HTTP"). This could, for example, integrate attestation and payments into the connection handshake and optimize message patterns for agent communication.

However, a clean-slate protocol would face significant deployment barriers: introducing new ALPN identifiers can trigger incompatible behavior in network middleboxes (firewalls, etc.), and it would require establishing new trust and certificate infrastructures for agent identities or attestations. It would likely see slow adoption, as all parties would need to upgrade to speak the new protocol.

By contrast, HAP takes a more incremental path by defining a profile within HTTP:

* It reuses HTTP/1.1, HTTP/2, and HTTP/3 transports and semantics unchanged.

* It adds profile-specific headers and behaviors (for authentication and payment) rather than altering core protocol syntax or introducing new methods.

* It can be implemented in middleware (reverse proxies, gateways) or libraries, meaning sites and agents can adopt it without a full-stack overhaul. An optional ALPN token or HTTP/2 setting can be used as a hint for efficiency but is not required for correctness.

We therefore focus on HAP an HTTP profile for authenticated, monetized agent traffic as the more practical approach. A future, purpose-built agent protocol (informally, "HTTPA") could be considered if experience with HAP reveals the need for deeper changes, but initially HAP stays within the bounds of HTTP to maximize deployability.

5. HTTP Agent Profile Overview

At a high level, the HTTP Agent Profile defines two conceptual lanes for web traffic:

1. _Human traffic lane:_ Requests from human-driven clients may carry a valid human token (e.g., a Privacy Pass token as per RFC 9578) when available. Such requests are treated as human-originated, meaning the server does not require them to be signed or paid through the HAP mechanisms. Human users thus continue to browse without additional protocol friction (aside from the invisible token exchange).

2. _Agent traffic lane:_ Requests from autonomous agents carry an HTTP Message Signature that binds the request to an agent identity (see Section 6). Servers recognizing these signatures can apply agent-specific policies: for example, issuing an HTTP 402 challenge to request payment, enforcing stricter rate limits, or denying access based on the agent's reputation or lack of compliance.

Requests with neither a signature nor a human token fall into a legacy or unknown category. Servers can handle these as they do today possibly with CAPTCHAs, blocking, or lower service quality or they may choose to gradually require HAP signals for certain resources once adoption is sufficient. The intention is that over time, most legitimate traffic will present one of the signals (human or agent), simplifying traffic classification.

6. Agent Authentication Using HTTP Message Signatures

HAP-compliant agents use HTTP Message Signatures ([RFC9421]) to authenticate their requests. Each HTTP request from an agent is augmented with a digital signature covering selected components of the request (such as headers and the request target).

Key establishment: An agent operator generates one or more cryptographic key pairs (for example, an Ed25519 key). The agent's public key(s) are published in a key directory for example at a known URL under the agent's control. A suggested convention is to use the HTTP Message Signatures key directory format at a well-known URI (for example, https://<agent-domain>/.well-known/agent-keys), which can list the agent's current public keys and metadata (key IDs, expiration, and so on).

Request structure: The agent includes the following headers in each request:

Signature-Input: Parameters indicating which parts of the HTTP request are covered by the signature, along with metadata like a creation time, expiration time, and a key identifier (kid). For example, it might indicate it signs the @authority (host), the @method and @path, and a custom Signature-Agent header (see below), with a creation timestamp and a key ID of "agent-key-2025-01".

Signature: The actual digital signature over the specified components, encoded in Base64. This is typically labeled (for example, "Signature: sig1=...") matching the label used in Signature-Input.

Signature-Agent: A header identifying the agent. This could be a URI (for example, "https://agent.example.com") or a name corresponding to the key directory. The contents of Signature-Agent are covered by the signature (by including "signature-agent" in the Signature-Input), ensuring an attacker cannot alter which agent is claiming to send the request.

Verification on the server: When a HAP-enabled server (or intermediary) receives a request with those headers, it will:

1. Parse the Signature-Input to understand which components were signed and obtain the key identifier (keyid) and any metadata (for example, timestamp).

2. Locate the agent's public key. This could involve fetching the key from the URL specified by Signature-Agent (if it is an HTTPS URI) or looking up a cached key if seen recently. The keyid helps select the correct key from the agent's key set.

3. Verify the signature using the public key and the reconstructed signing string (per [RFC9421]). If verification fails, the server knows the request was not actually from the claimed agent (or was tampered with), and it can be rejected (for example, treated as an invalid request or as a likely malicious bot).

4. If verification succeeds, the server now has an authenticated agent identity associated with the request. It can log this or use it in further policy decisions. For example, it might map the Signature-Agent URI to an internal identifier or check it against an allow/deny list.

Key rotation and metadata: The agent's key directory can indicate when keys expire or are revoked. Agents SHOULD rotate keys periodically (for instance, using a new keypair every 36 months) to limit the impact of key compromise. Servers SHOULD NOT cache keys beyond their advertised validity and SHOULD fetch updates as needed. If a previously seen key suddenly fails verification, the server MAY fetch a fresh copy of the directory in case of key rotation.

Impersonation resistance: Because the signature is tied to the agent's domain (or identifier), another party cannot masquerade as that agent without access to its private key. Even if a malicious bot claims the same Signature-Agent value as a well-known agent, its Signature header will not validate against the real agent's public key, and thus it will be recognized as a fake. This is a major improvement over the status quo where simply claiming User-Agent: Googlebot might fool some defenses.

7. Human vs. Agent Separation

HAP aims to avoid imposing protocol-level payment or signature requirements on human users whenever possible. For human-driven traffic, modern privacy-preserving human tokens such as Private Access Tokens [RFC9578] can provide a strong signal that a request originates from a human user on a legitimate device, without exposing a stable identifier.

In a HAP deployment, a server or intermediary can use the following classification strategy:

* If a request presents a valid human token, it is treated as human traffic. Normal user experience applies (ads, subscriptions, or site policies).

* If a request presents a valid agent signature, it is treated as agent traffic and subject to agent-specific policies (including possible payment requirements).

* If a request presents neither signal, it is treated as legacy or unknown traffic and handled according to existing bot mitigation and security policies.

User-side agents (such as browser-integrated assistants) MAY choose to present both a human token and an agent signature, to indicate that an autonomous component is acting on behalf of a specific user's browsing session. In such cases, servers can apply hybrid policies, for example permitting some forms of automated access at human-like rates without payment, while requiring payment for high-volume or bulk access.

8. HTTP 402 and Payment Profiles

HTTP status code 402 ("Payment Required") was reserved in early HTTP specifications but left without a standardized meaning for decades. HAP adopts 402 as a machine-readable signal that a request from an authenticated agent is potentially acceptable, but access is contingent on some form of payment or economic work.

When a server determines that an agent request requires payment, it responds with 402 instead of 200. The response includes sufficient information for the agent to understand how to fulfill the requirement. This information may appear in header fields (for example, a WWW-Authenticate challenge) and in the response body. Once the agent has fulfilled the requirement, it retries the request with proof of payment and, if validation succeeds, the server returns the requested resource.

HAP itself is payment-agnostic. It defines the use of 402 as a challenge mechanism and allows different payment systems to be bound to 402 via _payment profiles_. A payment profile specifies:

* How the server describes offers (price, currency, scope, and duration) in 402 responses.

* How the agent obtains a concrete payment request (for example, a Lightning invoice or other payment token).

* How the agent proves successful payment in subsequent requests.

One concrete payment profile is L402 ([L402]), which combines Macaroon tokens and the Lightning Network. Under L402, a server that wishes to charge an agent for access generates a Macaroon that encodes the scope and conditions of access, and a Lightning invoice for a specified amount. These are returned in a 402 response.

The agent then pays the Lightning invoice using its Lightning wallet or a custodial payment API. Upon successful payment, the agent obtains the invoice preimage. The agent retries the original request, including both the Macaroon (for example, in an Authorization header) and the preimage. The server verifies the Macaroon signature and caveats, and that the hash of the preimage matches the invoice hash encoded in the Macaroon. If both checks succeed, the server considers payment confirmed and grants access.

Other payment profiles could support different rails, such as on-chain cryptocurrency, fiat payment APIs, or proof-of-work-based puzzles. HAP does not mandate any particular payment rail; it simply provides the 402 envelope in which such profiles can operate for agent traffic.

9. Deployment and Operational Considerations

One of HAP's design goals is incremental deployability. Sites do not need to convert all APIs or endpoints to HAP at once, and existing clients are unaffected unless they choose to implement the profile.

In many deployments, reverse proxies, CDNs, and API gateways will be the primary enforcement points. These intermediaries can:

* Verify HTTP Message Signatures on incoming requests and classify them as agent or non-agent.

* Validate human tokens where available and route such requests down a human lane.

* Apply local policy for known agent identities (allow, rate limit, block, or require payment).

* Issue HTTP 402 challenges and interact with payment backends (Lightning nodes, payment gateways, or other services).

Origin servers can then be configured to see only requests that have either already satisfied HAP requirements or have been classified as non-HAP traffic. For example, an API gateway might only forward requests that present a valid agent signature and, where required, a proof of payment.

Servers and intermediaries can experiment with HAP policies in "report only" modes before enforcing them. For example, a proxy could log cases where an unauthenticated high-volume agent would have been challenged with 402, without actually issuing the challenge yet. This allows operators to tune thresholds and pricing before turning on enforcement.

As HAP use becomes more common, optional transport-level hints can improve efficiency. For example, a client might include an ALPN identifier in its TLS handshake to indicate HAP support, or an HTTP/2 SETTINGS parameter could advertise HAP capabilities. These optimizations are not required for correctness; all HAP functionality can be negotiated via normal HTTP messages.

10. Security Considerations

HAP introduces new security-relevant mechanisms at the HTTP layer. This section summarizes the main security considerations.

Authentication and impersonation. HAP relies on HTTP Message Signatures bound to an agent identifier for agent authentication. If an attacker obtains an agent's private key, it can impersonate that agent until keys are rotated and caches expire. Agent operators SHOULD protect private keys carefully, use hardware-backed storage where possible, and rotate keys regularly. Servers SHOULD respect key expiration metadata and avoid pinning keys for longer than necessary.

Replay and downgrade. Signatures SHOULD cover freshness indicators such as a timestamp and a nonce, and SHOULD be used over TLS. Servers SHOULD reject signatures that are too old or reuse the same nonce. Agents and servers SHOULD prefer HTTPS and avoid sending signed requests over cleartext HTTP.

Key discovery and SSRF. Key discovery via Signature-Agent and .well-known URLs introduces the risk of server-side request forgery (SSRF) or denial-of-service if an attacker can cause a server to fetch from arbitrary origins. Servers SHOULD restrict outbound key discovery to public IP ranges, enforce timeouts and connection limits, and cache keys according to HTTP caching rules to avoid repeated lookups.

Denial-of-service. Signature verification and payment processing consume CPU and network resources. Implementations SHOULD be designed to bound the amount of work per request, for example by limiting the number of keys tried for verification and by offloading payment processing to dedicated services. Operators SHOULD monitor for abuse patterns, such as floods of invalidly signed requests or repeated unpaid 402 challenges.

Abuse by authenticated agents. An authenticated and paying agent can still behave maliciously or violate a site's acceptable-use policy. HAP does not attempt to prevent misuse of content once access is granted. Instead, HAP aims to make agent traffic more accountable, so that misbehaving agent identities can be revoked or blocked and so that higher- level governance mechanisms and contracts can be applied.

11. Privacy Considerations

HAP affects privacy primarily through the introduction of agent identifiers and the handling of human tokens and payment metadata.

Agent identity and tracking. Persistent agent identifiers and long-lived keys can be used by servers to correlate agent behavior across sites. This is desirable for enterprise crawlers seeking to build reputation, but may be undesirable for user-centric agents that act on behalf of individuals. Operators of user-centric agents SHOULD use short-lived keys or per-origin keys to reduce the risk of cross-site tracking. Servers SHOULD avoid treating agent identities as direct proxies for user identities unless there is a separate, explicit user authentication relationship.

Human tokens. Privacy-preserving human tokens are designed so that issuers cannot link issuance and redemption events. HAP deployments should preserve this property by not logging or correlating human tokens beyond what is necessary for validation. Aggregated statistics (such as the rate of human versus agent traffic) can usually be collected without recording raw token values.

Payment metadata. Payment systems may reveal information about who pays for what. In many agent scenarios, payments are made by agent operators rather than end users, but care should be taken not to leak unnecessary payment metadata into application logs. Where possible, payment flows should be handled by dedicated payment services that expose only the minimal proof necessary for access control (for example, a validated token or flag).

12. IANA Considerations

This document does not currently define any new registries or request any actions from IANA. If future revisions define new HTTP header fields or authentication schemes specific to HAP, this section will be updated with appropriate registration requests.

13. Acknowledgements

The author thanks the broader HTTP, Web security, and AI standards communities for ongoing discussion about bot management, agent protocols, and monetization mechanisms, and acknowledges existing work on HTTP Message Signatures, Privacy Pass, and Lightning-based micropayments that helped shape this proposal.

14. 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>.

[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>.

[RFC9421] Backman, A., Ed., Richer, J., Ed., and M. Sporny, "HTTP Message Signatures", RFC 9421, DOI 10.17487/RFC9421, February 2024, <https://www.rfc-editor.org/info/rfc9421>.

[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>.

15. Informative References

[RFC9578] Celi, S., Davidson, A., Valdez, S., and C. A. Wood, "Privacy Pass Issuance Protocols", RFC 9578, DOI 10.17487/RFC9578, June 2024, <https://www.rfc-editor.org/info/rfc9578>.

[L402] Labs, L., "L402: Lightning HTTP 402 Protocol", 2023, <https://docs.l402.org/>.

Author’s Address

Sanat Dhir
Independent
Email: sdhir26@gsb.columbia.edu