TBD
Internet-Draft
Intended status: Standards Track
Expires: 4 October 2026

D. Hardt
Hell

2 April 2026

HTTP AAuth Headers
draft-hardt-aauth-headers-00

Abstract

This document defines two HTTP response headers — AAuth-Requirement and AAuth-Error — and profiles HTTP Message Signatures ([RFC9421]) for request authentication, with keying material conveyed via the Signature-Key header ([I-D.hardt-httpbis-signature-key]). A server uses AAuth-Requirement to require pseudonymous or verified agent identity, to request user interaction, or to signal that approval is pending. AAuth-Error conveys structured error information. Both headers use extensible registries for their values.

Discussion Venues

Note: This section is to be removed before publishing as an RFC.

This document is part of the AAuth specification family. Source for this draft and an issue tracker can be found at https://github.com/dickhardt/AAuth (https://github.com/dickhardt/AAuth).

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 4 October 2026.

Table of Contents

1. Introduction 3
2. Conventions and Definitions 3
3. Terminology 4
4. AAuth-Requirement HTTP Response Header 4
4.1. Header Structure 4
4.2. Requirement Levels 5
4.3. Pseudonym Required 5
4.4. Identity Required 6
4.5. Interaction Required 7
4.6. Approval Pending 9
5. HTTP Message Signatures Profile 9
5.1. Signature Algorithms 9
5.2. Keying Material 9
5.3. Signing (Agent) 10
5.3.1. Covered Components 10
5.3.2. Signature Parameters 10
5.4. Verification (Server) 10
6. AAuth-Error HTTP Response Header 11
6.1. Header Structure 11
6.2. Error Codes 12
6.2.1. invalid_request 12
6.2.2. invalid_input 12
6.2.3. invalid_signature 12
6.2.4. unsupported_algorithm 12
6.2.5. invalid_key 12
6.2.6. unknown_key 13
6.2.7. invalid_jwt 13
6.2.8. expired_jwt 13
6.3. Access Denied 13
7. Privacy Considerations 13
7.1. Key Thumbprint Tracking 13
7.2. Agent Identity Disclosure 14

7.3. JWKS Fetch Side Channel 14
8. Security Considerations 14
8.1. HTTP Message Signature Security 14
8.2. Replay Protection 14
8.3. JWKS Caching 15
9. IANA Considerations 15
9.1. HTTP Header Field Registration 15
9.2. AAuth Requirement Level Registry 15
9.3. AAuth Error Code Registry 16
10. Design Rationale 17
10.1. Why Not Extend WWW-Authenticate? 17
11. Implementation Status 17
12. Document History 18
13. Acknowledgments 18
14. References 18
14.1. Normative References 18
14.2. Informative References 19
Author's Address 20

1. Introduction

Modern distributed systems need a standard way for servers to communicate requirements to agents and for agents to present cryptographic identity. The existing HTTP WWW-Authenticate header ([RFC9110], Section 11.6.1) is limited to a fixed set of authentication schemes, cannot express progressive requirements, and cannot appear in 202 Accepted responses. The AAuth-Requirement header provides an extensible mechanism for progressive requirements — from pseudonymous access and verified identity to user interaction and deferred approval.

This specification defines:

* The AAuth-Requirement HTTP response header with an extensible requirement level registry
* Four requirement levels: pseudonym, identity, interaction, and approval
* A profile of HTTP Message Signatures ([RFC9421]) for request authentication, specifying required covered components, signature parameters, and keying material via the Signature-Key header ([I-D.hardt-httpbis-signature-key])
* The AAuth-Error HTTP response header with an extensible error code registry

2. Conventions and Definitions

{::boilerplate bcp14-tagged}

All HTTP requests and responses in AAuth MUST use HTTPS (HTTP over TLS ([RFC8446])). Any reference to "HTTP" in this document implies HTTPS unless explicitly noted otherwise.

3. Terminology

* *Agent*: An HTTP client ([RFC9110], Section 3.5) that signs its requests. In AAuth, agents have cryptographic identity.
* *Server*: An HTTP origin server ([RFC9110], Section 3.6) that communicates requirements via the AAuth-Requirement header.

4. AAuth-Requirement HTTP Response Header

Servers use the AAuth-Requirement response header to indicate requirements to agents. The header MAY be sent with 401 Unauthorized or 202 Accepted responses. A 401 response indicates that authentication or authorization is required. A 202 response indicates that the request is pending and additional action is required — either user interaction (requirement=interaction) or third-party approval (requirement=approval).

AAuth-Requirement and WWW-Authenticate are independent header fields; a response MAY include both. A client that understands AAuth processes AAuth-Requirement; a legacy client processes WWW-Authenticate. Neither header’s presence invalidates the other.

The header MAY also be sent with 402 Payment Required when a server requires both authentication and payment. The AAuth-Requirement conveys the authentication requirement; the payment requirement is conveyed by a separate mechanism such as x402 [x402] or the Micropayment Protocol (MPP) ([I-D.ryan-httpauth-payment]).

4.1. Header Structure

The AAuth-Requirement header field is a Dictionary ([RFC8941], Section 3.2). It MUST contain the following member:

* requirement: A Token ([RFC8941], Section 3.3.4) indicating the requirement level.

Additional members are defined per requirement level by the specification that registers the level. Recipients MUST ignore unknown members.

Example:

AAuth-Requirement: requirement=pseudonym

4.2. Requirement Levels

The requirement value is an extension point. This document defines four levels:

Level	Status Code	Meaning
pseudonym	401	Signed request proving key possession
identity	401	Verified agent identity
interaction	202	User action required at an interaction endpoint
approval	202	Approval pending, poll for result

Table 1

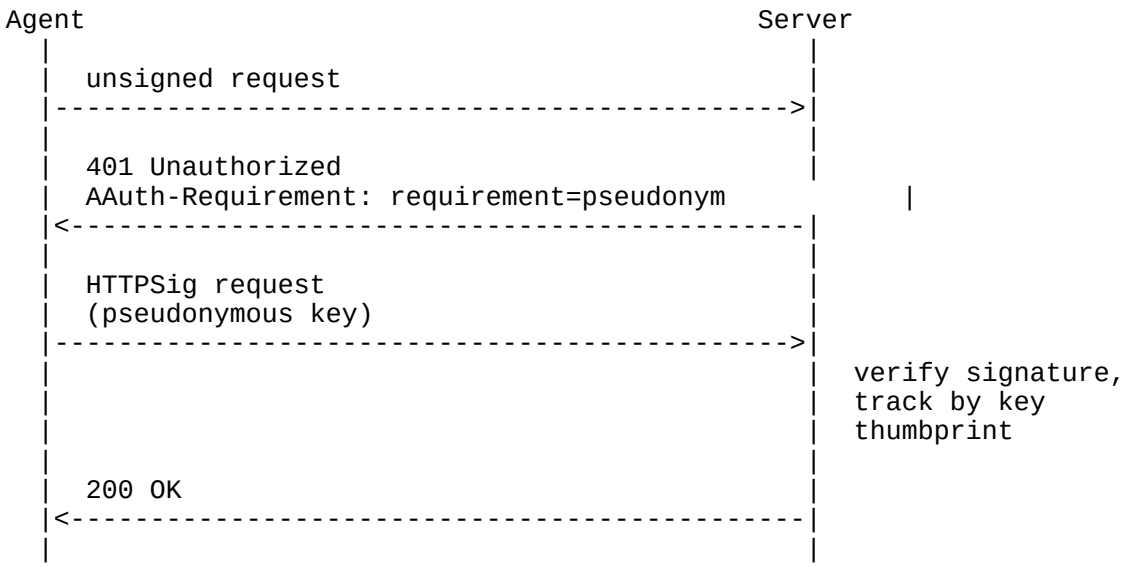
The AAuth Protocol specification (draft-hardt-aauth-protocol) registers additional levels (e.g., auth-token).

4.3. Pseudonym Required

When a server requires a signed request:

```
HTTP/1.1 401 Unauthorized
AAuth-Requirement: requirement=pseudonym
```

The agent retries with an HTTP Message Signature using a pseudonymous Signature-Key scheme ({{keying-material}}). The server can track the agent by JWK Thumbprint ([RFC7638]) without knowing its identity.



If the agent already knows the server requires pseudonymous access (from a previous interaction or metadata), it MAY sign the initial request directly without waiting for a 401 challenge.

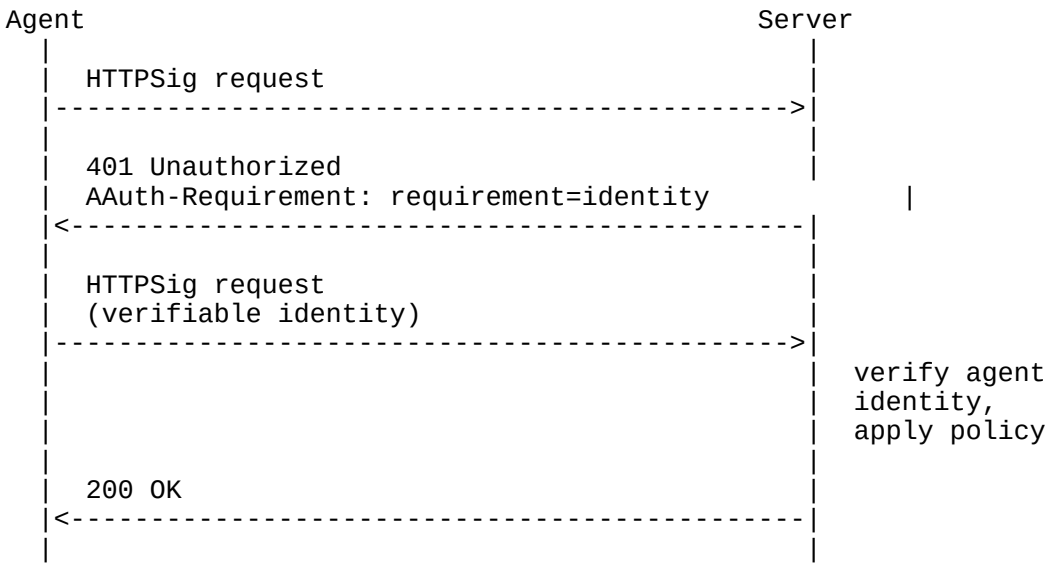
Use cases: Rate limiting anonymous requests, tracking repeat visitors by key thumbprint, spam prevention without requiring verified identity, hardware-backed pseudonymous identity.

4.4. Identity Required

When a server requires verified agent identity:

```
HTTP/1.1 401 Unauthorized
AAuth-Requirement: requirement=identity
```

The agent retries with a signed request using an identity Signature-Key scheme ({{keying-material}}).



If the agent already knows the server requires agent identity, it MAY present its identity on the initial request without waiting for a 401 challenge.

Use cases: API access policies based on known agents, webhook signature verification, allowlisting trusted agents for elevated rate limits.

4.5. Interaction Required

When a server requires user action — such as authentication, consent, payment approval, or any decision requiring a human in the loop — it returns a 202 Accepted response:

```
HTTP/1.1 202 Accepted
AAuth-Requirement: requirement=interaction; url="https://example.com/interact";
    code="A1B2-C3D4"
Location: /pending/f7a3b9c
Retry-After: 0
```

The AAuth-Requirement header MUST include the following parameters:

* url (String): The interaction URL where the user completes the required action. MUST use the https scheme and MUST NOT contain query or fragment components.
* code (String): An interaction code that links the agent's pending request to the user's session at the interaction URL.

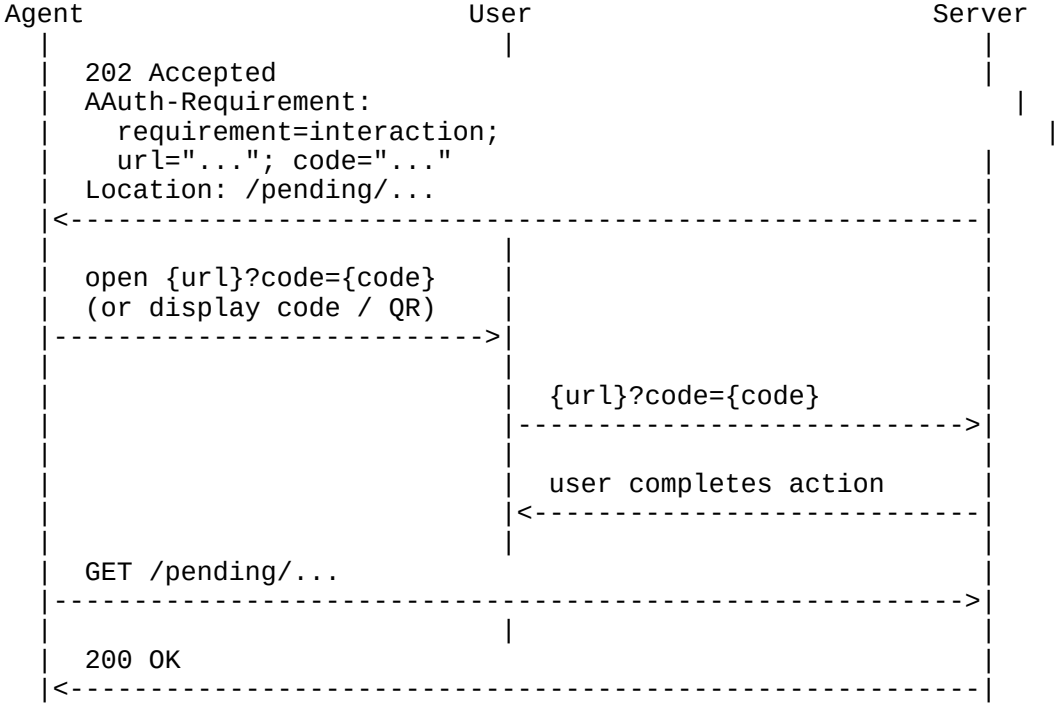
The response MUST also include:

* Location: A URL the agent polls (with GET) for a terminal response.
* Retry-After: Recommended polling interval in seconds.

The agent constructs a user-facing URL by appending the code as a query parameter: {url}?code={code}. The agent then directs the user to this URL using one of:

* *Browser redirect*: The agent opens the URL in the user's browser.
* *Display code*: The agent displays the url and code for the user to enter manually. The agent MAY also render the constructed URL as a QR code for the user to scan with their phone.

After directing the user, the agent polls the Location URL with GET requests, respecting the Retry-After interval. A 202 response means the request is still pending. A non-202 response is terminal — 200 indicates success, 403 indicates denial, and 408 indicates timeout.



Use cases: User login, consent, payment confirmation, document review, CAPTCHA, any workflow requiring human action.

4.6. Approval Pending

When a server is obtaining approval from another party without requiring the agent to direct a user — for example, via push notification, email, or administrator review:

```
HTTP/1.1 202 Accepted
AAuth-Requirement: requirement=approval
Location: /pending/f7a3b9c
Retry-After: 30
```

The response MUST include Location and Retry-After. The agent polls the Location URL with GET requests until a terminal response is received. No user action is required at the agent side. The same terminal response codes apply as for interaction.

Use cases: Administrator approval, resource owner consent, compliance review, direct user authorization via established communication channel.

5. HTTP Message Signatures Profile

This section profiles HTTP Message Signatures ([RFC9421]) for use with AAuth. Signing requirements (what the agent does) and verification requirements (what the server does) are specified separately.

5.1. Signature Algorithms

Agents and resources MUST support EdDSA using Ed25519 ([RFC8032]). Agents and resources SHOULD support ECDSA using P-256 with deterministic signatures ([RFC6979]). The alg parameter in the JWK ([RFC7517]) key representation identifies the algorithm. See the IANA JSON Web Signature and Encryption Algorithms registry ([RFC7518], Section 7.1) for the full list of algorithm identifiers.

5.2. Keying Material

The signing key is conveyed in the Signature-Key header ([I-D.hardt-httpbis-signature-key]). The Signature-Key scheme determines how the server obtains the public key:

* For pseudonym: the agent uses scheme=hwk (inline public key) or scheme=jkt-jwt (delegation from a hardware-backed key).
* For identity: the agent uses scheme=jwks_uri (JWKS endpoint) or scheme=jwt (JWT with public key in cnf claim).

See the Signature-Key specification ([I-D.hardt-httpbis-signature-key]) for scheme definitions, key discovery, and verification procedures.

5.3. Signing (Agent)

The agent creates an HTTP Message Signature ([RFC9421]) on each request, including the following headers:

* Signature-Key: Public key or key reference for signature verification
* Signature-Input: Signature metadata including covered components
* Signature: The HTTP message signature

5.3.1. Covered Components

The signature MUST cover the following derived components and header fields:

* @method: The HTTP request method ([RFC9421], Section 2.2.1)
* @authority: The target host ([RFC9421], Section 2.2.3)
* @path: The request path ([RFC9421], Section 2.2.6)
* signature-key: The Signature-Key header value

Servers MAY require additional covered components (e.g., content-digest ([RFC9530]) for request body integrity). The agent learns about additional requirements from server metadata or from an invalid_input error response that includes required_input ({{error-codes}}).

5.3.2. Signature Parameters

The Signature-Input header ([RFC9421], Section 4.1) MUST include the following parameters:

* created: Signature creation timestamp as an Integer (Unix time). The agent MUST set this to the current time.

5.4. Verification (Server)

When a server receives a signed request, it MUST perform the following steps. Any failure MUST result in a 401 response with the appropriate AAuth-Error header ({{error-codes}}).

1. Extract the Signature, Signature-Input, and Signature-Key headers. If any are missing, return invalid_request.

2. Verify that the Signature-Input covers the required components defined in {{covered-components}}. If the server requires additional components, verify those are covered as well. If not, return invalid_input with required_input.
3. Verify the created parameter is present and within 60 seconds of the server's current time. Reject with invalid_signature if outside this window. Servers and agents SHOULD synchronize their clocks using NTP ([RFC5905]).
4. Determine the signature algorithm from the alg parameter in the key. If the algorithm is not supported, return unsupported_algorithm.
5. Obtain the public key from the Signature-Key header according to the scheme, as specified in ([I-D.hardt-httpbis-signature-key]). Return invalid_key if the key cannot be parsed, unknown_key if the key is not found at the jwks_uri, invalid_jwt if a JWT scheme fails verification, or expired_jwt if the JWT has expired.
6. Verify the HTTP Message Signature ([RFC9421]) using the obtained public key and determined algorithm. Return invalid_signature if verification fails.

6. AAuth-Error HTTP Response Header

When a server rejects a request that includes AAuth signature headers (Signature, Signature-Input, and Signature-Key), the 401 response MUST include the AAuth-Error header.

6.1. Header Structure

The AAuth-Error header field is a Dictionary ([RFC8941], Section 3.2). It MUST contain the following member:

* error: A Token ([RFC8941], Section 3.3.4) indicating the error code.

Additional members are defined per error code. Recipients MUST ignore unknown members.

```
AAuth-Error: error=unsupported_algorithm,
    supported_algorithms=("EdDSA" "ES256")
```

The response body is OPTIONAL and MAY contain a human-readable description in any content type. The agent MUST NOT depend on the response body for error handling — all machine-readable error information is in the header.

6.2. Error Codes

6.2.1. invalid_request

The request is malformed or missing required information unrelated to signature verification — such as missing query parameters or an unsupported content type.

```
AAuth-Error: error=invalid_request
```

6.2.2. invalid_input

The Signature-Input is missing required covered components. The response SHOULD include a required_input member listing the components the server requires (see {{covered-components}} for the base set):

```
AAuth-Error: error=invalid_input,
    required_input=("@method" "@authority" "@path"
    "signature-key" "content-digest")
```

6.2.3. invalid_signature

The HTTP Message Signature is missing, malformed, or cryptographic verification failed. This includes missing Signature, Signature-Input, or Signature-Key headers, an expired created timestamp, or a signature that does not verify.

```
AAuth-Error: error=invalid_signature
```

6.2.4. unsupported_algorithm

The signing algorithm used by the agent is not supported by the server. The response MUST include a supported_algorithms member:

```
AAuth-Error: error=unsupported_algorithm,
    supported_algorithms=("EdDSA" "ES256")
```

6.2.5. invalid_key

The public key in Signature-Key could not be parsed, is expired, or does not meet the server’s trust requirements.

```
AAuth-Error: error=invalid_key
```

6.2.6. unknown_key

The public key from Signature-Key does not match any key at the agent's jwks_uri (applicable when the agent uses scheme=jwks_uri for verified identity; see {{keying-material}}). The server SHOULD re-fetch the JWKS once before returning this error, to handle key rotation.

AAuth-Error: error=unknown_key

6.2.7. invalid_jwt

The JWT in the Signature-Key header (when using scheme=jwt or scheme=jkt-jwt) is malformed or its signature verification failed.

AAuth-Error: error=invalid_jwt

6.2.8. expired_jwt

The JWT in the Signature-Key header (when using scheme=jwt or scheme=jkt-jwt) has expired (exp claim is in the past).

AAuth-Error: error=expired_jwt

6.3. Access Denied

When the server successfully verifies the agent's signature and identity but denies access based on policy (e.g., the agent is not authorized for this resource), the server returns 403 Forbidden. This is not an AAuth error — the authentication succeeded but authorization was denied. The response MUST NOT include an AAuth-Requirement or AAuth-Error header.

7. Privacy Considerations

7.1. Key Thumbprint Tracking

When an agent uses scheme=hwk (pseudonymous access), the server can track the agent across requests by JWK Thumbprint ([RFC7638]). If the agent uses the same key across multiple servers, those servers could correlate the agent's activity. Agents MUST use distinct keys for distinct servers to prevent cross-server correlation of pseudonymous identity.

7.2. Agent Identity Disclosure

When an agent presents its identity via scheme=jwks_uri or scheme=jwt, the server learns the agent's HTTPS URL. This reveals which software is making the request. Servers SHOULD NOT disclose agent identity information to third parties without the agent operator's consent.

7.3. JWKS Fetch Side Channel

When a server fetches an agent's JWKS from jwks_uri to verify a signature, the fetch itself reveals to the agent's JWKS host that someone is verifying signatures for that agent.

8. Security Considerations

8.1. HTTP Message Signature Security

HTTP Message Signatures provide:

1. *Request Integrity*: The signature covers HTTP method, target URI, and headers
2. *Replay Protection*: The created timestamp limits signature validity
3. *Key Binding*: Signatures are bound to specific keys via Signature-Key

8.2. Replay Protection

The 60-second validity window on the created timestamp (see {{verification-resource}}) limits the useful lifetime of a captured signature. The 60-second value balances clock-skew tolerance (NTP-synchronized hosts typically drift less than 10 seconds) against replay exposure — a shorter window would reject legitimate requests from hosts with modest clock drift, while a longer window would widen the replay attack surface.

Within that window, resources MUST maintain a cache of recently seen (key thumbprint, created) pairs and reject duplicate combinations. Without this cache, an attacker who captures a signed request can replay it within the validity window.

8.3. JWKS Caching

Resources that fetch JWKS documents for signature verification SHOULD cache the results with a TTL appropriate to their risk tolerance (recommended: 5 minutes for auth servers, 60 minutes for resources). JWKS endpoints SHOULD support standard HTTP caching headers (Cache-Control, Expires) per ([RFC9111]).

When signature verification fails due to an unknown key, the server SHOULD re-fetch the JWKS once before returning an unknown_key error, to handle key rotation.

9. IANA Considerations

9.1. HTTP Header Field Registration

This specification registers the following entry in the "Hypertext Transfer Protocol (HTTP) Field Name Registry" ([RFC9110], Section 16.3.1):

* Header Field Name: AAuth-Requirement
* Status: permanent
* Structured Type: Dictionary
* Reference: This document
* Header Field Name: AAuth-Error
* Status: permanent
* Structured Type: Dictionary
* Reference: This document

9.2. AAuth Requirement Level Registry

This specification establishes the AAuth Requirement Level Registry. The initial contents are:

Value	Reference
pseudonym	This document
identity	This document
interaction	This document
approval	This document

Table 2

New values may be registered following the Specification Required policy ([RFC8126]).

9.3. AAuth Error Code Registry

This specification establishes the AAuth Error Code Registry. The initial contents are:

Value	Reference
invalid_request	This document
invalid_input	This document
invalid_signature	This document
unsupported_algorithm	This document
invalid_key	This document
unknown_key	This document
invalid_jwt	This document
expired_jwt	This document

Table 3

New values may be registered following the Specification Required policy ([RFC8126]).

10. Design Rationale

10.1. Why Not Extend WWW-Authenticate?

WWW-Authenticate ([RFC9110], Section 11.6.1) tells the client which authentication scheme to use. Its challenge model is "present credentials" — it cannot express progressive requirements, authorization, or deferred approval, and it cannot appear in a 202 Accepted response.

AAuth-Requirement coexists with WWW-Authenticate. A 401 response MAY include both headers, and the client uses whichever it understands:

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Bearer realm="api"
AAuth-Requirement: requirement=identity
```

A 402 response MAY include WWW-Authenticate for payment (e.g., the Payment scheme defined by the Micropayment Protocol ([I-D.ryan-httpauth-payment])) alongside AAuth-Requirement for authentication or authorization:

```
HTTP/1.1 402 Payment Required
WWW-Authenticate: Payment id="x7Tg2pLq", method="example",
    request="eyJhbW91bnQiOiIxMDAw..."
AAuth-Requirement: requirement=pseudonym
```

11. Implementation Status

Note: This section is to be removed before publishing as an RFC.

This section records the status of known implementations of the protocol defined by this specification at the time of posting of this Internet-Draft, and is based on a proposal described in [RFC7942]. The description of implementations in this section is intended to assist the IETF in its decision processes in progressing drafts to RFCs.

The following implementations are known at the time of writing:

* *@aauth npm packages* (https://www.npmjs.com/org/aauth): (https://www.npmjs.com/org/aauth):) JavaScript/TypeScript libraries implementing HTTP Message Signatures and AAuth-Requirement processing for agents and servers.

* *aauth-implementation* (https://github.com/christian-posta/aauth-implementation): (https://github.com/christian-posta/aauth-implementation):) Python library implementing HTTP Message Signatures (RFC 9421), AAuth request signing/verification, and Signature-Key header support. Author: Christian Posta.

12. Document History

Note: This section is to be removed before publishing as an RFC.

* draft-hardt-aauth-headers-00
  - Initial submission, renamed from draft-hardt-aauth-header
  - Added AAuth-Error header with extensible error code registry
  - Added interaction and approval requirement levels
  - Added url parameter to interaction for self-contained challenge
  - Renamed terminology from "resource" to "server"

13. Acknowledgments

TBD

14. References

14.1. Normative References

[I-D.hardt-httpbis-signature-key]
Hardt, D. and T. Meunier, "HTTP Signature-Key Header", Work in Progress, Internet-Draft, draft-hardt-httpbis-signature-key-02, 2 March 2026, <https://datatracker.ietf.org/doc/html/draft-hardt-httpbis-signature-key-02>.

[I-D.ryan-httpauth-payment]
Ryan, B., Moxey, J., Meagher, T., Weinstein, J., and S. Kaliski, "The "Payment" HTTP Authentication Scheme", Work in Progress, Internet-Draft, draft-ryan-httpauth-payment-01, 17 March 2026, <https://datatracker.ietf.org/doc/html/draft-ryan-httpauth-payment-01>.

[RFC6979] Pornin, T., "Deterministic Usage of the Digital Signature Algorithm (DSA) and Elliptic Curve Digital Signature Algorithm (ECDSA)", RFC 6979, DOI 10.17487/RFC6979, August 2013, <https://www.rfc-editor.org/info/rfc6979>.

[RFC7517] Jones, M., "JSON Web Key (JWK)", RFC 7517, DOI 10.17487/RFC7517, May 2015, <https://www.rfc-editor.org/info/rfc7517>.

[RFC7638] Jones, M. and N. Sakimura, "JSON Web Key (JWK) Thumbprint", RFC 7638, DOI 10.17487/RFC7638, September 2015, <https://www.rfc-editor.org/info/rfc7638>.

[RFC8032] Josefsson, S. and I. Liusvaara, "Edwards-Curve Digital Signature Algorithm (EdDSA)", RFC 8032, DOI 10.17487/RFC8032, January 2017, <https://www.rfc-editor.org/info/rfc8032>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/info/rfc8126>.

[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>.

[RFC8941] Nottingham, M. and P. Kamp, "Structured Field Values for HTTP", RFC 8941, DOI 10.17487/RFC8941, February 2021, <https://www.rfc-editor.org/info/rfc8941>.

[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>.

[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>.

14.2. 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>.

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

[RFC7942] Sheffer, Y. and A. Farrel, "Improving Awareness of Running Code: The Implementation Status Section", BCP 205, RFC 7942, DOI 10.17487/RFC7942, July 2016, <https://www.rfc-editor.org/info/rfc7942>.

[RFC9111] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Caching", STD 98, RFC 9111, DOI 10.17487/RFC9111, June 2022, <https://www.rfc-editor.org/info/rfc9111>.

[RFC9530] Polli, R. and L. Pardue, "Digest Fields", RFC 9530, DOI 10.17487/RFC9530, February 2024, <https://www.rfc-editor.org/info/rfc9530>.

[x402] x402 Foundation, "x402: HTTP 402 Payment Protocol", 2025, <https://docs.x402.org>.

Author's Address

Dick Hardt
Hell
Email: dick.hardt@gmail.com