Web Bot Auth
Internet-Draft
Intended status: Informational
Expires: 23 April 2026

T. Meunier
Cloudflare

20 October 2025

HTTP Message Signatures for automated traffic Architecture
draft-meunier-web-bot-auth-architecture-04

Abstract

This document describes an architecture for identifying automated traffic using [HTTP-MESSAGE-SIGNATURES]. The goal is to allow automated HTTP clients to cryptographically sign outbound requests, allowing HTTP servers to verify their identity with confidence.

About This Document

This note is to be removed before publishing as an RFC.

The latest revision of this draft can be found at https://thibmeu.github.io/http-message-signatures-directory/draft-meunier-web-bot-auth-architecture.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-meunier-web-bot-auth-architecture/.

Discussion of this document takes place on the Web Bot Auth Working Group mailing list (mailto:web-bot-auth@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/web-bot-auth/. Subscribe at https://www.ietf.org/mailman/listinfo/web-bot-auth/.

Source for this draft and an issue tracker can be found at https://github.com/thibmeu/http-message-signatures-directory.

Status of This Memo

This Internet-Draft will expire on 23 April 2026.

Table of Contents

1. Introduction

Agents are increasingly used in business and user workflows, including AI assistants, search indexing, content aggregation, and automated testing. These agents need to reliably identify themselves to origins for several reasons:

1. Regulatory compliance requiring transparency of automated systems

2. Origin resource management and access control

3. Protection against impersonation and reputation management

4. Service level differentiation between human and automated traffic

Current identification methods such as IP allowlisting, User-Agent strings, or shared API keys have significant limitations in security, scalability, and manageability. This document defines an architecture enabling agents to cryptographically identify themselves using [HTTP-MESSAGE-SIGNATURES]. It proposes that every request from bots to be signed by a private key owned by its provider. This way, every origin can validate the service identity.

2. Motivation

There is an increase in agent traffic on the Internet. Many agents choose to identify their traffic today via IP Address lists and/or unique User-Agents. This is often done to demonstrate trust and safety claims, support allowlisting/denylisting the traffic in a granular manor, and enable sites to monitor and rate limit per agent operator. However, these mechanisms have drawbacks:

1. User-Agent, when used alone, can be spoofed meaning anyone may attempt to act as that agent. It is also overloaded - an agent may be using Chromium and wish to present itself as such to ensure rendering works, yet it still wants to differentiate its traffic to the site.

2. IP blocks alone can present a confusing story. IPs on cloud plaforms have layers of ownership - the platform owns the IP and registers it in their published IP blocks, only to be re-published by the agent with little to bind the publication to the actual service provider that may be renting infra. Purchasing dedicated IP blocks is expensive, time consuming, and requires significant specialist knowledge to set up. These IP blocks may have prior reputation history that needs to be carefully inspected and managed before purchase and use.

3. An agent may go to every website on the Internet and share a secret with them like a Bearer from [OAUTH-BEARER]. This is impractical to scale for any agent beyond select partnerships, and insecure, as key rotation is challenging and becomes less secure as the consumers scale.

Using well-established cryptography, we can instead define a simple and secure mechanism that empowers small and large agents to share their identity.

2.1. HTTP layer choice

This architecture operates solely at the HTTP layer. It allows signatures to be generated and verified without modifying the transport layer or TLS stack. It enables flexible deployment across proxies, gateways, and origin servers, and aligns with existing tooling and infrastructure that already inspect and manipulate HTTP headers.

Because the signature is embedded in the request itself, it travels with the message through intermediaries, preserving end-to-end verifiability even when requests are forwarded or transformed within the HTTP layer.

3. Conventions and Definitions

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.

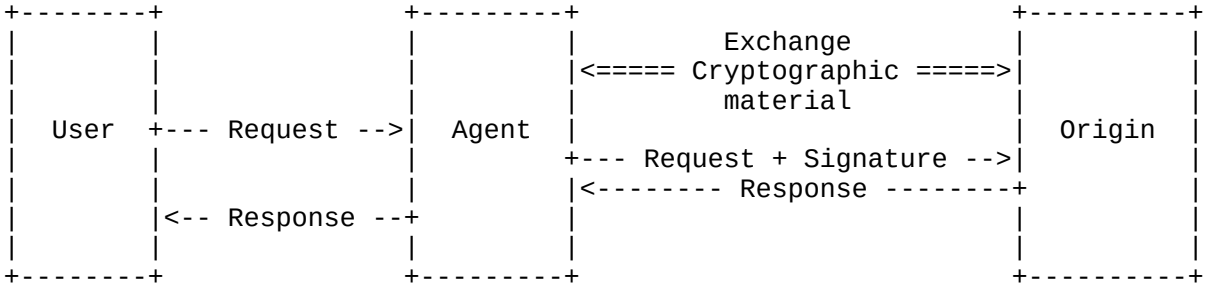
The following terms are used throughout this document:

User An entity initiating requests through an agent. May be a human operator or another system.

Agent An orchestrated user agent (e.g. Chromium, CURL). It implements the HTTP protocol and constructs valid HTTP requests with [HTTP-MESSAGE-SIGNATURES] signatures.

Origin An HTTP server receiving signed requests that implements the HTTP protocol and verifies [HTTP-MESSAGE-SIGNATURES] signatures. It acts as a verifier of the signature as defined by [HTTP-MESSAGE-SIGNATURES].

4. Architecture



A User initiates an action requiring the Agent to perform an HTTP request. The Agent constructs the request, generates a signature using its signing key, and includes it in the request as defined in Section 3.1 of [HTTP-MESSAGE-SIGNATURES] along with the Signature-Agent header for discovery for its verification key. Upon receiving the request, the Origin ensures it has the verification key for the Agent, validates the signature, and processes the request if the signature is valid.

4.1. Deployment Models

Signature verification can be performed either directly by origins or delegated to a fronting proxy. Direct verification by origins provides simplicity and control. Proxy verification offloads processing and enables shared caching across multiple origins. The choice depends on traffic volume and operational requirements.

4.2. Generating HTTP Message Signature

[HTTP-MESSAGE-SIGNATURES] defines components to be signed.

Agents MUST include at least one of the following components:

@authority as defined in Section 2.2.3 of [HTTP-MESSAGE-SIGNATURES]

@target-uri as defined in Section 2.2.2 of [HTTP-MESSAGE-SIGNATURES]

Agents MUST include the following @signature-params as defined in Section 2.3 of [HTTP-MESSAGE-SIGNATURES]

created as defined in Section 2.3 of [HTTP-MESSAGE-SIGNATURES]

expires as defined in Section 2.3 of [HTTP-MESSAGE-SIGNATURES]

keyid MUST be a base64url JWK SHA-256 Thumbprint as defined in Section 3.2 of [JWK-THUMBPRINT] for RSA and EC, and in Appendix A.3 of [JWK-OKP] for ed25519.

tag MUST be web-bot-auth

The signing key is available to the agent at request time. Algorithms should be registered with IANA as part of HTTP Message Signatures Algorithm registry.

The creation of the signature is defined in Section 3.1 of [HTTP-MESSAGE-SIGNATURES].

It is RECOMMENDED the expiry to be no more than 24 hours.

4.2.1. Signature-Agent

Signature-Agent is an HTTP Method context header defined in Section 4.1 of [DIRECTORY]. It is RECOMMENDED that the Agent sends requests with Signature-Agent header, as described in Section 4.2.4. If the header is to be sent, one of its members MUST be signed as a component as defined in Section 2.1 of [HTTP-MESSAGE-SIGNATURES].

This results in the following components to be signed

("@authority" "signature-agent";key="sig1")

It is RECOMMENDED that the key matches the signature label.

4.2.2. Anti-replay

Origins MAY want to prevent signatures from being spoofed or used multiple times by bad actors and thus require a nonce to be added to the @signature-params. This is described in Section 7.2.2 of [HTTP-MESSAGE-SIGNATURES].

Agents SHOULD extend @signature-parameters defined in Section 4.2 as follows:

nonce base64url encoded random byte array. It is RECOMMENDED to use a 64-byte array.

Client MUST ensure that this nonce is unique for the validity window of the signature, as defined by created and expires attributes.

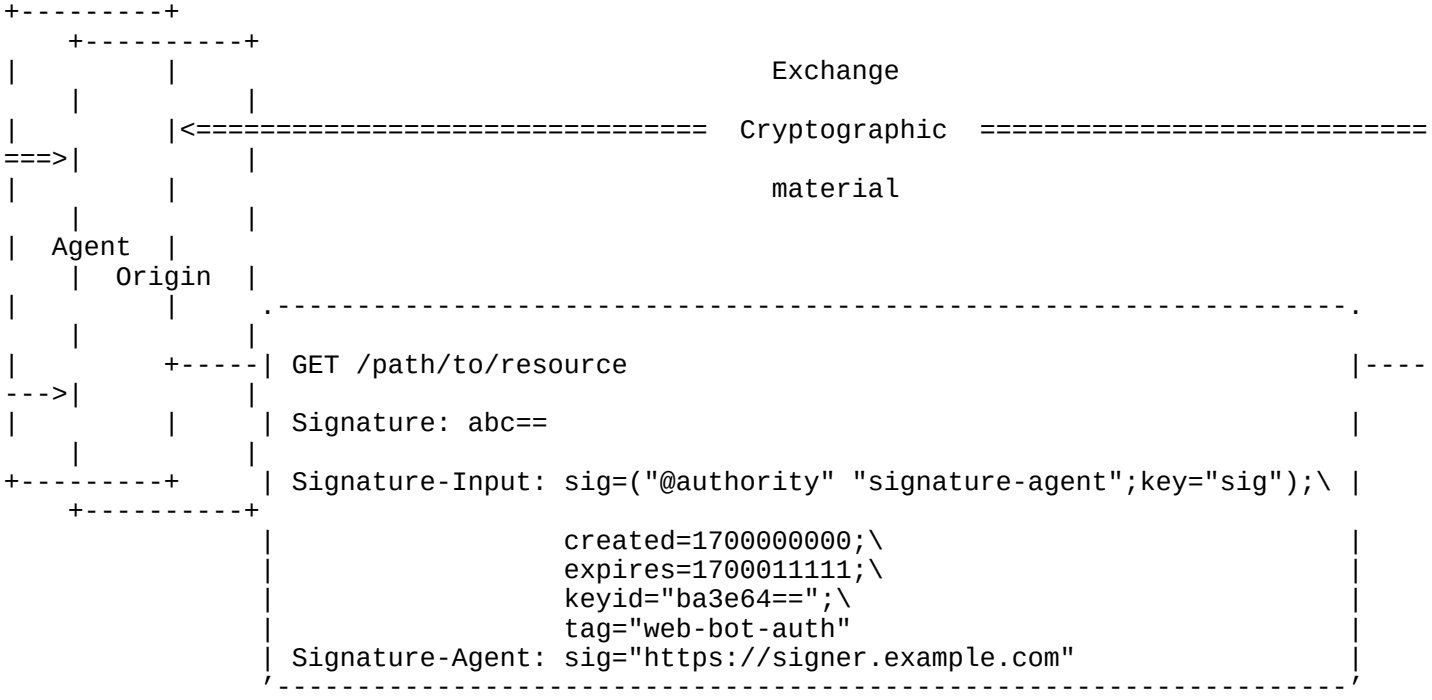
4.2.3. Additional headers

Agents MAY include additional components, such as specific HTTP headers, in the signature. This can be prompted by the origin requesting additional headers, as described in Section 4.3, or initiated by the agent to provide more information within the signature scope. For example, an agent might include an HTTP header expressing its intent and sign it.

Origins MAY ignore certain headers at their own discretion, and request a new signature, as described in Section 4.3.

4.2.4. Sending a request

An Agent SHOULD send a request with the signature generated above. Updating the architecture diagram, the flow looks as follow.



The Agent SHOULD send requests with two headers

1. Signature defined in Section 4.2

2. Signature-Input defined in Section 4.2

Mentioned in Section 4.2.1, the Agent MAY send requests with Signature-Agent header.

4.3. Requesting a Message signature

Section 5 of [HTTP-MESSAGE-SIGNATURES] defines the Accept-Signature field which can be used to request a Message Signature from a client by an origin. An Origin MAY choose to request signatures from clients that did not initially provide them. If requesting, Origins MUST use the same parameters as those defined by the Section 4.2. The status code SHOULD be 403 Forbidden as defined in Section 15.5.4 of [HTTP].

Origin MAY request a new signature with tag "web-bot-auth" even if a nonce is provided, for example if it believes the nonce is a replay, or if it doesn't store nonces and thus requests new signatures every time. The status code SHOULD be 429 Too Many Requests as defined in Section 4 of [HTTP-MORE-STATUS-CODE].

4.4. Validating Message signature

Upon receiving an HTTP request, the origin has to verify the signature. The algorithm is provided in Section 3.2 of [HTTP-MESSAGE-SIGNATURES]. Similar to a regular User-Agent check, this happens at the HTTP layer, once headers are received.

Additional requirements are placed on this validation:

* During step 1 to 3 included, if the Origin fails to parse the provided Signature, Signature-Input, or Signature-Agent headers, it MAY respond with status code 400 Bad Request as defined in Section 15.5.1 of [HTTP].

* During step 4, the Origin MAY discard signatures for which the tag is not set to web-bot-auth.

* During step 5, the Origin MAY discard signatures for which they do not know the keyid.

* During step 5, if the keyid is unknown to the origin, they MAY fetch the provider directory as indicated by Signature-Agent header defined in Section 4 of [DIRECTORY].

Origin MAY require the nonce to satisfy certain constraint: be globally unique using a global nonce store, be unique to a specific location or time window using a local cache, or not constraint at all.

4.5. Key Distribution and Discovery

This section describes the discovery mechanism for the agent directory.

The reference for discovery is a FQDN. It SHOULD provide a directory hosted on the well known registered in Section 4 of [DIRECTORY].

Example:

```
{
  "keys": {
    "kty": "OKP",
    "crv": "Ed25519",
    "kid": "NFcWBst6DXG-N35nHdzMrioWntdzNZghQSkjHNMMSjw",
    "x": "JrQLj5P_89iXES9-vFgrIy29clF9CC_oPPsw3c5D0bs",
    "use": "sig",
    "nbf": 1712793600,
    "exp": 1715385600
  }
}
```

4.5.1. Out-of-band communication between client and origin

A service submitting their key to an origin, or the origin manually adding a service to their trusted list.

4.5.2. Public list

Could be a GitHub repository like the public suffix list. The issue is the gating of such repositories, and therefore its governance.

4.5.3. Signature-Agent header

This allows for backward compatibility with existing header agent filtering, and an upgrade to a cryptographically secured protocol. See Section 4.2.1 for more details.

5. Security Considerations

5.1. Use of TLS

We reassess Section 7.1.2 of [HTTP-MESSAGE-SIGNATURES]. Clients SHOULD use TLS [RFC8446] (https) or equivalent transport security when making requests with Message signatures. Failing to do so exposes the Message signature to numerous attacks that could give attackers unintended access.

This include reverse proxy and their consideration presented in Section 5.7.

An origin SHOULD refuse Signature headers when communicated over an unsecured channel.

5.2. Performance Impact

Origins should account for the overhead of signature verification in their operations. A local cache of public keys reduces network requests and verification latency. The choice of signing algorithm impacts CPU requirements. Origins should monitor verification latency and set appropriate timeouts to maintain service levels under load.

5.3. Nonce validation

Clients are the one controlling the nonce. While Section 4.2.2 mandates that clients MUST provide a globally unique nonce, it is the origin’s responsibility to enforce it.

Different validation policies have different performance and operational considerations. Global uniqueness requires a global nonce store. Some origins may find that their use case can tolerate sharding on location, timing, or other properties.

5.4. Key Compromise Response

An agent signing key might get compromised.

If that happens, the agent SHOULD cease using the compromised key as soon as possible, notify affected origins if possible, and generate a new key pair.

To minimise the impact of a key compromise, the origin should support rapid key rotation and monitor for suspicious signature patterns.

5.5. Shared Secrets Considered Harmful

Implementations MUST NOT use shared HMAC defined in Section 3.3.3 of [HTTP-MESSAGE-SIGNATURES]. Shared secrets break non-repudiation and make auditing difficult. Each automated client SHOULD use a unique asymmetric keypair to ensure attribution, support key rotation, and enable effective revocation if needed.

5.6. Key Reuse Considered Harmful

Implementations SHOULD NOT reuse a signing key for different purposes. For example, if an agent implementor has two agents they want to differentiate, these should use distinct signing keys and signing key directories.

5.7. Reverse proxy consideration

An origin may be placed behind a reverse proxy, which means the proxy will see the Signature and Signature-Agent headers before the origin does. A proxy SHOULD NOT strip the Signature or Signature-Agent headers from requests.

A proxy SHOULD NOT replay signatures against other reverse proxies used by the origin, as this allows impersonation of the principal signature agent.

Origins MAY require a specific nonce policy to prevent such malicious behaviour and decide to validate the signature themselves. This has to be done in accordance with Section 5.3. For example, an origin could require a nonce derived from public information (such as the current date), mandate nonce chaining (where each nonce is the hash of the previous one), or provide its own nonce in an Accept-Signature response to challenge the agent.

Such policies MAY incur additional round-trip between the client and the origin to convey accept-signature header, or deployment specific exchanges.

5.7.1. Signature-Agent labeling

An intermediary is allowed to relabel an existing signature when processing the message, per Section 7.2.5 of [HTTP-MESSAGE-SIGNATURES].

This MAY apply to Signature-Agent, when included in the request as defined in Section 4.2.1, An intermediary updating the member key MUST update the components of the associated signatures accordingly.

For instance, an intermediary updating the Signature-Agent from sig2 to sig3 on the example provided in Appendix A.1.2 would result in the following Signature, Signature-Input, and Signature-Agent header.

```
NOTE: '\' line wrapping per RFC 8792

Signature-Agent: sig3="https://signature-agent.test"
Signature-Input: sig2=("@authority" "signature-agent";key="sig3")\
 ;created=1735689600\
 ;keyid="oD0HwocPBSfpNy5W3bpJeyFGY_IQ_YpqxSjQ3Yd-CLA"\
 ;alg="rsa-pss-sha512"\
 ;expires=1735693200\
 ;nonce="XSHtZVCThSIAksXsH9WBs6AtxtXC0eQGiIcUGSoJstFs8lAWakjhrfwzLhyjtme5iXMZvmFWqDEs6cT3
Jf+BbQ=="\
 ;tag="web-bot-auth"
Signature: sig2=:I1QWNzGXdP1a4dSvOHLCVOOanEYHDk+ZsVxM9MLX/p4ko69ghKwR5EOtAD96g7g4GWP7lmpM
/jFAf9q8EFRDTPLjUXySwMv4YPgabv2LQihTJG2y8a2m6IGltyruwQNiqSJVUuRaG9+b17CGmAMFZh30X6GXLdQJr
CARpeTqPwp2DC+a8haDE/VE5EruqzjA5/2mKwvrkzkSqeW5tOVtFwWRRHIOidquf/8Je6kM9mhgkg4arudLA5SL4w
yyYE1jURIgcOl8agrfdJ5Def23DIRtiOLRa8jT9cpTLFAuFHN+mrZA/LH9h0gSIg1cPb+0cMASee5uku1KjWcFer7
jWA==:
```

Signature is unchanged as the base is similar. Both Signature-Agent and Signature-Input reflect the update from sig2 to sig3. . # Privacy Considerations

5.8. Public Identity

This architecture assumes that automated clients identify themselves explicitly using digital signatures. The identity associated with a signing key is expected to be publicly discoverable for verification purposes. This reduces anonymity and allows receivers to associate requests with specific agents. If an agent wishes not to identify itself, this is not the right choice of protocol for it.

5.9. No Human Correlation

The key used for signing MUST NOT be tied to a specific human individual. Keys SHOULD represent a role, company, or automation identity (e.g., "news-aggregator- bot", "example-crawler-v1"). This avoids accidental exposure of personally identifiable information and prevents the misuse of keys for user tracking or profiling.

5.10. Minimizing Tracking Risks

To limit tracking risks, implementations SHOULD avoid long-lived, globally unique key identifiers unless strictly necessary. Key rotation SHOULD be supported, and clients SHOULD take care to avoid signing information that could be used to correlate activity across contexts, especially where sensitive user data is involved.

6. IANA Considerations

This document has no IANA actions.

7. References

7.1. Normative References

[DIRECTORY] Meunier, T., "HTTP Message Signatures Directory", Work in Progress, Internet-Draft, draft-meunier-http-message-signatures-directory-04, 20 October 2025, <https://datatracker.ietf.org/doc/html/draft-meunier-http-message-signatures-directory-04>.

[HTTP] 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/rfc/rfc9110>.

[HTTP-CACHE] 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/rfc/rfc9111>.

[HTTP-MESSAGE-SIGNATURES] 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/rfc/rfc9421>.

[HTTP-MORE-STATUS-CODE] Nottingham, M. and R. Fielding, "Additional HTTP Status Codes", RFC 6585, DOI 10.17487/RFC6585, April 2012, <https://www.rfc-editor.org/rfc/rfc6585>.

[JWK-OKP] Liusvaara, I., "CFRG Elliptic Curve Diffie-Hellman (ECDH) and Signatures in JSON Object Signing and Encryption (JOSE)", RFC 8037, DOI 10.17487/RFC8037, January 2017, <https://www.rfc-editor.org/rfc/rfc8037>.

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

[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/rfc/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/rfc/rfc8174>.

7.2. Informative References

[OAUTH-BEARER]
Jones, M. and D. Hardt, "The OAuth 2.0 Authorization Framework: Bearer Token Usage", RFC 6750, DOI 10.17487/RFC6750, October 2012, <https://www.rfc-editor.org/rfc/rfc6750>.

[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/rfc/rfc8446>.

Appendix A. Test Vectors

A.1. RSASSA-PSS Using SHA-512

The test vectors in this section use the RSA-PSS key defined in Appendix B.1.2 of [HTTP-MESSAGE-SIGNATURES]. This section includes non-normative test vectors that may be used as test cases to validate implementation correctness.

A.1.1. Signature-Agent absent from the request

This example presents a minimal signature using the rsa-pss-sha512 algorithm over test-request. The request does not contain a Signature-Agent header.

The corresponding signature base is:

NOTE: '\' line wrapping per RFC 8792

```
"@authority": example.com
"@signature-params": ("@authority")\
 ;created=1735689600\
 ;keyid="oD0HwocPBSfpNy5W3bpJeyFGY_IQ_YpqxSjQ3Yd-CLA"\
 ;alg="rsa-pss-sha512"\
 ;expires=1735693200\
 ;nonce="yT+sZR1glKOTemVLbmPDFwPScbB1Zj/sMNPEFZcjwJW5jK/taa7HviOXovVwiZOfrrLHS2SbLFUQBxPY
ZChf7g=="\
 ;tag="web-bot-auth"
```

This results in the following Signature-Input and Signature header fields being added to the message under the label sig1:

NOTE: '\' line wrapping per RFC 8792

```
Signature-Input: sig1=("@authority")\
 ;created=1735689600\
 ;keyid="oD0HwocPBSfpNy5W3bpJeyFGY_IQ_YpqxSjQ3Yd-CLA"\
 ;alg="rsa-pss-sha512"\
 ;expires=1735693200\
 ;nonce="yT+sZR1glKOTemVLbmPDFwPScbB1Zj/sMNPEFZcjwJW5jK/taa7HviOXovVwiZOfrrLHS2SbLFUQBxPY
ZChf7g=="\
 ;tag="web-bot-auth"
Signature: sig1=:ppXhcGjVp7xaoHGIa7V+hsSxuRgFt8i04K4FWz9ORJtn57t8duD3cyavsnh9grdWWOJHER8I
TNBaqe4mKmPq193S+7hSW31IzXSH4/9WfsdrjUBwyJ0fhBU7oNn3UGDqwdhr5TMgVI2/EX8saV5GrOunM09zMEA+d
4QWYyKRFJmg+asCs253l2IYPpVp4N55H0uRK7qhb7acng8LNiEPTQZD2s+Kha95LgeciKQSO0jgR/h59fX/dXqLdF
IvRMn8Ggs2VUzF/f/MMEXH83gufVnh4SYl/rKMSKWDBsK+OiLpobAVIuIz+HLCVlMnxlkXkhCW2J/Pmo8jht9N5k/
y1A==:
```

A.1.2. Signature-Agent included present on the request

This example presents a minimal signature using the rsa-pss-sha512 algorithm over test-request. The request contains a Signature-Agent header.

The corresponding signature base is:

NOTE: '\' line wrapping per RFC 8792

```
"@authority": example.com
"signature-agent": sig2="https://signature-agent.test"
"@signature-params": ("@authority" "signature-agent";key="sig2")\
 ;created=1735689600\
 ;keyid="oD0HwocPBSfpNy5W3bpJeyFGY_IQ_YpqxSjQ3Yd-CLA"\
 ;alg="rsa-pss-sha512"\
 ;expires=1735693200\
 ;nonce="XSHtZVCThSIAksXsH9WBs6AtxtXC0eQGiIcUGSoJstFs8lAWakjhrfwzLhyjtme5iXMZvmFWqDEs6cT3
Jf+BbQ=="\
 ;tag="web-bot-auth"
```

This results in the following Signature-Input and Signature header fields being added to the message under the label sig2:

NOTE: '\' line wrapping per RFC 8792

```
Signature-Agent: sig2="https://signature-agent.test"
Signature-Input: sig2=("@authority" "signature-agent";key="sig2")\
 ;created=1735689600\
 ;keyid="oD0HwocPBSfpNy5W3bpJeyFGY_IQ_YpqxSjQ3Yd-CLA"\
 ;alg="rsa-pss-sha512"\
 ;expires=1735693200\
 ;nonce="XSHtZVCThSIAksXsH9WBs6AtxtXC0eQGiIcUGSoJstFs8lAWakjhrfwzLhyjtme5iXMZvmFWqDEs6cT3
Jf+BbQ==="\
 ;tag="web-bot-auth"
Signature: sig2=:I1QWNzGXdP1a4dSvOHLCVOOanEYHDk+ZsVxM9MLX/p4ko69ghKwR5EOtAD96g7g4GWP7lmpM
/jFAf9q8EFRDTPLjUXySwMv4YPgabv2LQihTJG2y8a2m6IGltyruwQNiqSJVUuRaG9+b17CGmAMFZh30X6GXLdQJr
CARpeTqPwp2DC+a8haDE/VE5EruqzjA5/2mKwvrkzkSqeW5tOVtFwWRRHIOidquf/8Je6kM9mhgkg4arudLA5SL4w
yyYE1jURIgcOl8agrfdJ5Def23DIRtiOLRa8jT9cpTLFAuFHN+mrZA/LH9h0gSIg1cPb+0cMASee5uku1KjWcFer7
jWA==:
```

A.2. EdDSA Using Curve edwards25519

The test vectors in this section use the Ed25519 key defined in Appendix B.1.4 of [HTTP-MESSAGE-SIGNATURES]. This section include non-normative test vectors that may be used as test cases to validate implementation correctness.

A.2.1. Signature-Agent absent from the request

This example presents a minimal signature using the ed25519 algorithm over test-request. The request does not contain a Signature-Agent header.

The corresponding signature base is:

NOTE: '\' line wrapping per RFC 8792

```
"@authority": example.com
"@signature-params": ("@authority")\
 ;created=1735689600\
 ;keyid="poqkLGiymh_W0uP6PZFw-dvez3QJT5SolqXBCW38r0U"\
 ;alg="ed25519"\
 ;expires=1735693200\
 ;nonce="mYotfW3CUjI68sbGw6oKd7kyXqPjZEtU8xFPGWFrqOAf5qC6MDe3pys3SWWCudB0MvwslHy32WXUpkR7
u0lt/w=="\
 ;tag="web-bot-auth"
```

This results in the following Signature-Input and Signature header fields being added to the message under the label sig1:

NOTE: '\' line wrapping per RFC 8792

```
Signature-Input: sig1=("@authority")\
 ;created=1735689600\
 ;keyid="poqkLGiymh_W0uP6PZFw-dvez3QJT5SolqXBCW38r0U"\
 ;alg="ed25519"\
 ;expires=1735693200\
 ;nonce="mYotfW3CUjI68sbGw6oKd7kyXqPjZEtU8xFPGWFrqOAf5qC6MDe3pys3SWWCudB0MvwslHy32WXUpkR7
u0lt/w=="\
 ;tag="web-bot-auth"
Signature: sig1=:+NA/cssf4Y2bQTMTkyvTGRCaVzp9quyUevdwwMtMOWhhOOZ2T1subBj0BtvdnrpDEuwSAbiT
eElXDzHL3WWKCw==:
```

A.2.2. Signature-Agent included present on the request

This example presents a minimal signature using the ed25519 algorithm over test-request. The request contains a Signature-Agent header.

The corresponding signature base is:

NOTE: '\' line wrapping per RFC 8792

```
"@authority": example.com
"signature-agent": sig2="https://signature-agent.test"
"@signature-params": ("@authority" "signature-agent";key="sig2")\
 ;created=1735689600\
 ;keyid="poqkLGiymh_W0uP6PZFw-dvez3QJT5SolqXBCW38r0U"\
 ;alg="ed25519"\
 ;expires=1735693200\
 ;nonce="e8N7S2MFd/qrd6T2R3tdfAuuANngKI7LFtKYI/vowzk4lAZYadIX6wW25MwG7DCT9RUKAJ0qVkU0mEeL
ElW1qg=="\
 ;tag="web-bot-auth"
```

This results in the following Signature-Input and Signature header fields being added to the message under the label sig2:

NOTE: '\' line wrapping per RFC 8792

```
Signature-Agent: sig2="https://signature-agent.test"
Signature-Input: sig2=("@authority" "signature-agent";key="sig2")\
 ;created=1735689600\
 ;keyid="poqkLGiymh_W0uP6PZFw-dvez3QJT5SolqXBCW38r0U"\
 ;alg="ed25519"\
 ;expires=1735693200\
 ;nonce="e8N7S2MFd/qrd6T2R3tdfAuuANngKI7LFtKYI/vowzk4lAZYadIX6wW25MwG7DCT9RUKAJ0qVkU0mEeL
ElW1qg==\
 ;tag="web-bot-auth"
Signature: sig2=:jdq0SqOwHdyHr9+r5jw3iYZH6aNGKijYp/EstF4RQTQdi5N5YYKrD+mCT1HA1nZDsi6nJKuH
xUi/5Syp3rLWBA==:
```

Appendix B. Implementations

This draft has a couple of public implementations. A demonstration server has been deployed to https://http-message-signatures-example.research.cloudflare.com/ (https://http-message-signatures-example.research.cloudflare.com/).

It uses ed25519 example signing and verifying keys defined in Appendix B.1.4 of [HTTP-MESSAGE-SIGNATURES].

B.1. Clients

* Chrome MV3 (https://github.com/cloudflare/web-bot-auth) (TypeScript)
* Cloudflare Workers (https://github.com/cloudflare/web-bot-auth) (TypeScript)
* Puppeteer script (https://github.com/stytchauth/web-bot-auth-example) (JavaScript)
* Guzzle middleware (https://github.com/olipayne/guzzle-web-bot-auth-middleware) (PHP)
* Python script (https://zenn.dev/oymk/articles/944069e5eddc27) (Python)
* Bot-Authentication (https://github.com/cyberstormdotmu/bot-authentication) (Python)
* HTPie plugin (https://github.com/cloudflare/web-bot-auth) (Python)
* Web scrapers (scrapy/crawl4ai) (https://github.com/cyberstormdotmu/bot-authentication) (Python)
* HUMAN Verified AI Agents (https://github.com/HumanSecurity/human-verified-ai-agent) (Python)
* Linzer (https://github.com/nomadium/linzer/blob/master/spec/integration/cloudflare_example_research_spec.rb) (Ruby)
* Rust binaries (https://github.com/cloudflare/web-bot-auth) (Rust)

B.2. Servers

* Caddy plugin (https://github.com/cloudflare/web-bot-auth) (Go)

* Cloudflare Workers (https://github.com/cloudflare/web-bot-auth) (TypeScript)

* Apache module (https://github.com/garyillyes/web-bot-auth-apache) (C)

B.3. Test vectors

* In JSON format (https://github.com/cloudflare/web-bot-auth/blob/main/packages/web-bot-auth/test/test_data/web_bot_auth_architecture_v1.json)

Acknowledgments

The editor would also like to thank the following individuals (listed in alphabetical order) for feedback, insight, and implementation of this document - Marwan Fayed, Maxime Guerreiro, Scott Hendrickson, Jonathan Hoyland, Alex Sandor Major, Mark Nottingham, Eugenio Panero, Lucas Pardue, Malte Ubl, Loganaden Velvindron, Tanya Verma.

Changelog

v04

* Change Signature-Agent to a sf-dictionary

* Add security consideration around intermediary re-keying Signature-Agent

* Add target-uri as a valid replacement for @authority

* Add more contributors

* Add new implementations of architecture drafts

* Remove "purpose" field from web-bot-auth architecture example

v03

* Update linzer example url

* Fix section and name of status code 429

* Fix typos

v02

* Add response status code

* Add a few reference to improve readability
* Add consideration to sign additional headers
* Add use of TLS in security considerations
* Add RSASSA-PSS example
* Update acknowledgement section
* Reference new implementations: PHP, Python, Ruby, Rust
* Fix signature-agent in the arch diagram to use structured fields
* Fix test vectors to use signature-agent with structured fields
* Fix some typos

v01

* Add mandatory signing of Signature-Agent by clients if present
* Add test vectors for request with and without Signature-Agent
* Update example diagram to be correct
* Add security consideration about reverse proxy
* Update why origin may request a new signature
* Update nonce validation wording and global uniqueness
* Acknowledgements

v00

* Initial draft
* How to leverage HTTP Message Signature to sign request
* How to verify these Signature
* Define web-bot-auth tag to scope this signature
* Derive keyid using JWK Thumbprint
* High level Security and Privacy considerations

Author's Address

Thibault Meunier
Cloudflare
Email: ot-ietf@thibault.uk