HTTP
Internet-Draft
Intended status: Standards Track
Expires: 12 July 2026

D. Hardt
Hell
T. Meunier
Cloudflare

8 January 2026

HTTP Signature-Key Header
draft-hardt-httpbis-signature-key-01

Abstract

This document defines the Signature-Key HTTP header field for distributing public keys used to verify HTTP Message Signatures as defined in RFC 9421. Four initial key distribution schemes are defined: pseudonymous inline keys (hwk), identified signers with JWKS URI discovery (jwks_uri), X.509 certificate chains (x509), and JWT-based delegation (jwt). These schemes enable flexible trust models ranging from privacy-preserving pseudonymous verification to PKI-based identity chains and horizontally-scalable delegated authentication.

Discussion Venues

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

Source for this draft and an issue tracker can be found at https://github.com/dickhardt/signature-key (https://github.com/dickhardt/signature-key).

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 12 July 2026.

Table of Contents

1. Introduction

HTTP Message Signatures [RFC9421] provides a powerful mechanism for creating and verifying digital signatures over HTTP messages. To verify a signature, the verifier needs the signer’s public key. While RFC 9421 defines signature creation and verification procedures, it intentionally leaves key distribution to application protocols, recognizing that different deployments have different trust requirements.

This document defines the Signature-Key HTTP header field to standardize key distribution for HTTP Message Signatures. The header enables signers to provide their public key or a reference to it directly in the HTTP message, allowing verifiers to obtain keying material without prior coordination.

The header supports four schemes, each designed for different trust models and operational requirements:

1. *Header Web Key (hwk)* - Self-contained public keys for pseudonymous verification
2. *JWKS URI (jwks_uri)* - Identified signers with key discovery via metadata
3. *X.509 (x509)* - Certificate-based verification with PKI trust chains
4. *JWT (jwt)* - Delegated keys embedded in signed JWTs for horizontal scale

Additional schemes may be defined through the IANA registry established by this document.

The Signature-Key header works in conjunction with the Signature-Input and Signature headers defined in RFC 9421, using matching labels to correlate signature metadata with keying material.

2. The Signature-Key Header Field

The Signature-Key header field provides the public key or key reference needed to verify an HTTP Message Signature. The header is a Structured Field Dictionary [RFC8941] keyed by signature label, where each member describes how to obtain the verification key for the corresponding signature.

Format:

Signature-Key: <label>=<scheme>;<parameters>...

Where: - <label> (dictionary key) matches the label in Signature-Input and Signature headers - <scheme> (token) identifies the key distribution scheme - <parameters> are semicolon-separated key-value pairs where values are strings that vary by scheme

Multiple keys are comma-separated per the dictionary format. See [RFC8941] for definitions of dictionary, token, and string.

Example:

```
Signature-Input: sig=("@method" "@authority" "@path" "signature-key"); created=1732210000
Signature: sig=:MEQCIA5...
Signature-Key: sig=hwk;kty="OKP";crv="Ed25519";x="JrQLj..."
```

Label Correlation:

Labels are correlated by equality of label names across Signature-Input, Signature, and Signature-Key. Signature-Key is a dictionary keyed by label; Signature-Input and Signature are the sources of what signatures are present; Signature-Key provides keying material for those labels.

Verifiers MUST:

1. Parse Signature-Input and Signature per RFC 9421 and obtain the set of signature labels present. The verifier determines which labels it is attempting to verify based on application context and RFC 9421 processing.

2. Parse Signature-Key as a Structured Fields Dictionary

3. For each label being verified, select the Signature-Key dictionary member with the same name

4. If the corresponding dictionary member is missing, verification for that signature MUST fail

| *Note:* A verifier might choose to verify only a subset of labels present (e.g., the application-required signature); labels not verified can be ignored.

2.1. Label Consistency

If a label appears in Signature or Signature-Input, and the verifier attempts to verify it, the corresponding member MUST exist in Signature-Key. If Signature-Key contains members for labels not being verified, verifiers MAY ignore them.

2.2. Multiple Signatures

The dictionary format supports multiple signatures per message. Each signature has its own dictionary member keyed by its unique label:

```
Signature-Input: sig1=(... "signature-key"), sig2=(... "signature-key")
Signature: sig1=:...:, sig2=:...:
Signature-Key: sig1=jwt;jwt="eyJ...", sig2=jwks_uri;id="https://example.com";well-known="
meta";kid="k1"
```

Most deployments SHOULD use a single signature. When multiple signatures are required, the complete Signature-Key header (containing all keys) MUST be populated before any signature is created, and each signature MUST cover signature-key. This ensures all signatures protect the integrity of all key material. See Signature-Key Integrity (#signature-key-integrity) in Security Considerations. Alternative key distribution mechanisms outside this specification may be used for scenarios requiring independent signature addition.

2.3. Header Web Key (hwk)

The hwk scheme provides a self-contained public key inline in the header, enabling pseudonymous verification without key discovery.

Parameters by key type:

OKP (Octet Key Pair):

* kty (REQUIRED) - "OKP"
* crv (REQUIRED) - Curve name (e.g., "Ed25519")
* x (REQUIRED) - Public key value

```
Signature-Key: sig=hwk;kty="OKP";crv="Ed25519";x="JrQLj5P..."
```

EC (Elliptic Curve):

* kty (REQUIRED) - "EC"
* crv (REQUIRED) - Curve name (e.g., "P-256", "P-384")
* x (REQUIRED) - X coordinate
* y (REQUIRED) - Y coordinate

```
Signature-Key: sig=hwk;kty="EC";crv="P-256";x="f83OJ3D...";y="x_FEzRu..."
```

RSA:

* kty (REQUIRED) - "RSA"

* n (REQUIRED) - Modulus

* e (REQUIRED) - Exponent

```
Signature-Key: sig=hwk;kty="RSA";n="0vx7agoebGcQ...";e="AQAB"
```

Constraints:

* The alg parameter MUST NOT be present (algorithm is specified in Signature-Input)

* The kid parameter SHOULD NOT be used

Use cases:

* Privacy-preserving agents that avoid identity disclosure

* Experimental or temporary access without registration

* Rate limiting and reputation building on a per-key basis

2.4. JWKS URI Discovery (jwks_uri)

The jwks_uri scheme identifies the signer and enables key discovery via a metadata document containing a jwks_uri property.

Parameters:

* id (REQUIRED) - Signer identifier (HTTPS URL)

* well-known (REQUIRED) - Metadata document name under /.well-known/

* kid (REQUIRED) - Key identifier

> *Note:* The well-known parameter may be shortened to wk once the semantics are stable.

Discovery procedure:

1. Fetch {id}/.well-known/{well-known}

2. Parse as JSON metadata

3. Extract jwks_uri property

4. Fetch JWKS from jwks_uri

5. Find key with matching kid

Example:

```
Signature-Key: sig=jwks_uri;id="https://agent.example";well-known="aauth-agent";kid="key-1"
```

Use cases:

* Identified services with stable HTTPS identity

* Search engine crawlers and monitoring services

* Services requiring explicit entity identification

2.5. X.509 Certificates (x509)

The x509 scheme provides certificate-based verification using PKI trust chains.

Parameters:

* x5u (REQUIRED) - URL to X.509 certificate chain (PEM format, [RFC7517] Section 4.6)

* x5t (REQUIRED) - Certificate thumbprint: BASE64URL(SHA256(DER(leaf_cert)))

Verification procedure:

1. Check cache for certificate with matching x5t

2. If not cached or expired, fetch PEM from x5u

3. Validate certificate chain to trusted root CA

4. Check certificate validity and revocation status

5. Verify x5t matches leaf certificate

6. Extract public key from end-entity certificate

7. Verify signature using extracted key

8. Cache certificate indexed by x5t

Example:

```
Signature-Key: sig=x509;x5u="https://agent.example/.well-known/cert.pem";x5t="bWcoon4QTVn
8Q6xiY0ekMD6L8bNLMkuDV2KtvsFc1nM"
```

Use cases:

* Enterprise environments with PKI infrastructure

* Integration with existing certificate management systems

* Scenarios requiring certificate revocation checking

* Regulated industries requiring certificate-based authentication

2.6. JWT Confirmation Key (jwt)

The jwt scheme embeds a public key inside a signed JWT using the cnf (confirmation) claim [RFC7800], enabling delegation and horizontal scale.

Parameters:

* jwt (REQUIRED) - Compact-serialized JWT

JWT requirements:

* MUST contain cnf.jwk claim with embedded JWK

* SHOULD contain standard claims: iss, sub, exp, iat

| *Note:* The mechanism by which the JWT is obtained is out of scope
| of this specification.

Verification procedure:

1. Validate the JWT: verify signature using issuer’s public key and verify claims per policy (iss, exp, etc.)

2. Extract JWK from cnf.jwk

3. Verify HTTP Message Signature using extracted key

Example:

```
Signature-Key: sig=jwt;jwt="eyJhbGciOiJFUzI1NiI..."
```

JWT payload example:

```
{
  "iss": "https://issuer.example",
  "sub": "instance-123",
  "exp": 1732210000,
  "cnf": {
    "jwk": {
      "kty": "OKP",
      "crv": "Ed25519",
      "x": "JrQLj5P_89iXES9-vFgrIy29clF9CC_oPPsw3c5D0bs"
    }
  }
}
```

Use cases:

* Distributed services with ephemeral instance keys

* Delegation scenarios where instances act on behalf of an authority

* Short-lived credentials for horizontal scaling

3. Security Considerations

3.1. Key Validation

Verifiers MUST validate all cryptographic material before use:

* *hwk*: Validate JWK structure and key parameters

* *jwks_uri*: Verify HTTPS transport and validate fetched JWKS

* *x509*: Validate complete certificate chain, check revocation status

* *jwt*: Verify JWT signature and validate embedded JWK

3.2. Caching and Performance

Verifiers MAY cache keys to improve performance but MUST implement appropriate cache expiration:

* *jwks_uri*: Respect cache-control headers, implement reasonable TTLs

* *x509*: Cache by x5t, invalidate on certificate expiry

* *jwt*: Cache embedded keys until JWT expiration

Verifiers SHOULD implement cache limits to prevent resource exhaustion attacks.

3.3. Scheme-Specific Risks

hwk: No identity verification - suitable only for scenarios where pseudonymous access is acceptable.

jwks_uri: Relies on HTTPS security - vulnerable to DNS/CA compromise. Verifiers should implement certificate pinning where appropriate.

x509: Requires robust certificate validation including revocation checking. Verifiers MUST NOT skip certificate chain validation.

jwt: Delegation trust depends on JWT issuer verification. Verifiers MUST validate JWT signatures and claims before trusting embedded keys.

3.4. Algorithm Selection

The alg parameter in Signature-Input (RFC 9421) determines the signature algorithm. Verifiers MUST:

* Validate algorithm against policy (reject weak algorithms)

* Ensure key type matches algorithm requirements

* Reject algorithm/key mismatches

3.5. Signature-Key Integrity

The Signature-Key header SHOULD be included as a covered component in Signature-Input:

```
Signature-Input: sig=("@method" "@authority" "@path" "signature-key"); created=1732210000
```

If signature-key is not covered, an attacker can modify the header without invalidating the signature. Attacks include:

Scheme substitution: An attacker extracts the public key from an hwk scheme and republishes it via jwks_uri under their own identity, causing verifiers to attribute the request to the attacker.

Identity substitution: An attacker modifies the id parameter in a jwks_uri scheme to point to their own metadata endpoint that returns the same public key, impersonating a different signer.

Verifiers SHOULD reject requests where signature-key is not a covered component.

4. Privacy Considerations

4.1. Pseudonymity vs. Identity

The hwk scheme enables pseudonymous operation where the signer's identity is not disclosed. Verifiers should be aware that:

* hwk provides no identity linkage across requests (unless keys are reused)
* Key reuse enables tracking but may be necessary for reputation/rate-limiting
* Verifiers should not log or retain hwk keys beyond operational necessity

The jwks_uri, x509, and jwt schemes all reveal signer identity. Protocols using these schemes should inform signers that their identity will be disclosed to verifiers.

4.2. Key Discovery Tracking

The jwks_uri and x509 schemes require verifiers to fetch resources from signer-controlled URLs. This creates potential tracking vectors:

* Signers can observe when and from where keys are fetched
* Verifiers should cache keys to minimize fetches
* Verifiers may wish to use shared caching infrastructure to reduce fingerprinting

4.3. JWT Contents

JWTs in the jwt scheme may contain additional claims beyond cnf. Verifiers should:

* Only process claims necessary for verification
* Not log or retain unnecessary JWT claims
* Be aware that JWT contents are visible to network observers unless using TLS

5. IANA Considerations

5.1. HTTP Field Name Registration

This document registers the Signature-Key header field in the "Hypertext Transfer Protocol (HTTP) Field Name Registry" defined in [RFC9110].

Header field name: Signature-Key

Applicable protocol: http

Status: standard

Author/Change controller: IETF

Specification document(s): [this document]

5.2. Signature-Key Scheme Registry

This document establishes the "HTTP Signature-Key Scheme" registry. This registry allows for the definition of additional key distribution schemes beyond those defined in this document.

5.2.1. Registration Procedure

New scheme registrations require Specification Required per [RFC8126].

5.2.2. Initial Registry Contents

Scheme	Description	Reference
hwk	Header Web Key - inline public key	[this document]
jwks_uri	JWKS URI Discovery - key discovery via metadata	[this document]
x509	X.509 Certificate - PKI certificate chain	[this document]
jwt	JWT Confirmation Key - delegated key in JWT	[this document]

Table 1

5.2.3. Registration Template

Scheme Name: The token value used in the Signature-Key header
Description: A brief description of the scheme
Specification: Reference to the specification defining the scheme
Parameters: List of parameters defined for this scheme

6. Normative References

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

[RFC7800] Jones, M., Bradley, J., and H. Tschofenig, "Proof-of-Possession Key Semantics for JSON Web Tokens (JWTs)", RFC 7800, DOI 10.17487/RFC7800, April 2016, <https://www.rfc-editor.org/info/rfc7800>.

[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>.

[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>.

Appendix A. Acknowledgments

The author would like to thank reviewers for their feedback on this specification.

Authors' Addresses

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

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