HTTP Signature Keys

draft-hardt-httpbis-signature-key-07

Dick Hardt (Hellō) & Thibault Meunier (Cloudflare)

httpbis — IETF 126 Vienna — July 2026

Agenda

1. Quick overview of Signature-Key
2. What's happened since IETF 125
   a. `dwk` parameter for `.well-known` value
   b. New schemes: `jkt-jwt`, `self-jwt`
   c. New mechanisms: `sigkey` parameter, `Signature-Error`
   d. Implementations, adoption & demos
3. **The bigger picture: client authentication for HTTP**

Quick Overview: The Missing Piece

- HTTP Message Signatures (RFC 9421) — proof of possession at the application layer
  - works through proxies and CDNs, unlike mTLS
- To verify a signature, the verifier needs the signer's **public key**
- RFC 9421 intentionally leaves **key distribution** to application protocols
- **Signature-Key** fills that gap: a header that carries the key — or where to find it — inline with the signed message

```
Signature-Input: sig=("@method" "@authority" "@path" "signature-key"); created=1784246000
Signature:       sig=:MEQCIA5...
Signature-Key:   sig=hwk;kty="OKP";crv="Ed25519";x="JrQLj..."
```

Schemes for Different Trust Models

Scheme	Model	Identity
hwk	Pseudonymous	Key thumbprint
jkt-jwt	Pseudonymous + delegation	Stable enclave key thumbprint
jwks_uri	Identified	HTTPS URL + well-known metadata
self-jwt	Identified + claims	Issuer is the signer
jwt	Identified + delegated + claims	JWT issuer signs for instances
x509	PKI	Certificate subject

The scheme tells the verifier the **trust model** and **verification procedure** — extensible via IANA registry

Scheme Parameters

Scheme	Parameters
hwk	JWK inline:
	OKP `kty` ; `crv` ; `x` — EC `kty` ; `crv` ; `x` ; `y` — RSA `kty` ; `n` ; `e`
jkt-jwt	`jwt` — enclave key in JWT header `jwk` , ephemeral key in `cnf` (RFC 7800)
jwks_uri	`id` (HTTPS URL); `dwk` (well-known doc); `kid` (key identifier)
self-jwt	`jwt` — `iss` + `dwk` claims, `kid` header, no `cnf`
jwt	`jwt` — key in `cnf.jwk` , issuer discovery via `iss` + `dwk`
x509	`x5u` (cert chain URL); `x5t` (SHA-256 cert thumbprint)

Since IETF 125: Five Revisions

Draft	Date	Highlights
-03	Apr 4	`jkt-jwt` scheme, `dwk` parameter, early validation
-04	Apr 9	Renamed **"HTTP Signature Keys"**; `sigkey` parameter; Signature-Error header
-05	Jun 17	Implementer feedback — SSRF defenses, caching rules
-06	Jul 2	`self-jwt` scheme
-07	Jul 4	Editorial; AAuth & Email Verification cited as users

`dwk` — Generic Key Discovery via .well-known

Problem: every protocol defines its own `.well-known` metadata document

- `openid-configuration`, `oauth-authorization-server`, `aauth-resource`, …
- Verifier would need to know *which* document each application uses — per-application code just to verify a signature

Solution: the signed request names its own metadata document

- `dwk` ("dot well-known") — a parameter in the `jwks_uri` scheme or a JWT claim (`jwt`, `self-jwt` schemes)
- Verifier fetches `{id or iss}/.well-known/{dwk}` and extracts `jwks_uri`

A generic verifier can verify any signed request — **no understanding of the `.well-known` file required** — while applications reuse metadata they already deploy

New Scheme: jkt-jwt (from preview → shipped)

Previewed at IETF 125, now fully specified in -03:

- Hardware enclave key (slow, secure) delegates to an **ephemeral software key** (fast) via a self-issued JWT
- Stable identity: `urn:jkt:sha-256:<thumbprint>` (JWK Thumbprint URI)
- TOFU trust model (RFC 7435) — hardware-rooted identity, software signing speed

New Scheme: `self-jwt` (-06)

- JWT **issuer and HTTP signer are the same party** — no `cnf` claim
- One key, discovered via `{iss}/.well-known/{dwk}`, verifies both JWT and HTTP signature
- Carries claims (`sub`, `aud`, …) alongside an identified signature

New Mechanism: `sigkey` Parameter for Accept-Signature

Server signals the *type* of key it requires (RFC 9421 §5):

- `jkt` (pseudonymous: `hwk`, `jkt-jwt`)
- `uri` (identified: `jwks_uri`, `jwt`, `self-jwt`, `x509` with URI SAN)
- `x509` (PKI: `x509`)

New Mechanism: `Signature-Error`

Response header with structured verification errors:

- `invalid_signature`, `unknown_key`, `expired_jwt`, …
- Extensible with IANA error code registry

Incremental Adoption — Zero Coordination

Servers can adopt signatures **without breaking existing clients**:

1. Server responds `429` / `401` / `402` with `Accept-Signature: sig=();sigkey=jkt`
2. Client retries with a signed request and Signature-Key
3. Verification failures return **Signature-Error** so clients can self-diagnose

- No pre-registration, no out-of-band setup
- Unsigned traffic keeps working — signed traffic gets better treatment
- Rate limit by key thumbprint or issuer instead of by IP

Specs Building on Signature-Key

- **AAuth** (draft-hardt-oauth-aauth-protocol) — all parties use Signature-Key to distribute the keys that verify their signed requests
- **Email Verification** (draft-hardt-email-verification, DISPATCH this week) — uses `hwk` to convey the browser's public key so the issuer can bind it into the verification token

Demos: Replacing API Keys with Signature Keys

Lightning demos at **AAuth Night** (San Francisco, July 1, 2026):

Demo	Who
Vestauth — auth for agents, from the creator of `dotenv` / `dotenvx`	Scott Motte
Keycard — identity and access platform for AI agents	Jared Hanson
LoginID — strong authentication for agents acting for users	Jesse Ariss
MailChannels — email sending for AI agents	Ken Simpson
AAuth Web Agent — live AAuth protocol demo	Dick Hardt

Signature-Key enables moving from **API Keys** to **durable identifiers**

Hardened by Implementation

Feedback from Joshua Gay, implementing the draft in **sidecat**, drove -05:

- **Egress admission checklist** for `jwks_uri` fetches:
  HTTPS only, size/timeout limits, redirect policy, reject private/loopback addresses, DNS rebinding defense
- **Caching rules**: once-per-minute fetch floor; one refresh-and-retry on same-`kid` signature failure before returning `unknown_key`

The spec is being shaped by people building it, not just reviewing it

The Bigger Picture: Client Identity Is Unsolved

Server identity — solved

- DNS + TLS + Let's Encrypt: globally unique names, cryptographic binding, no gatekeeper

Client identity — not solved

- mTLS never escaped the enterprise perimeter
- On the open web, every client falls back to **shared secrets**: API keys, passwords

API keys are bearer tokens — whoever has the key *is* the client

- 39M secrets leaked on GitHub in 2024 (GitHub); 90% still valid 5 days later (GitGuardian)

The Same Primitives Solve It Everywhere

Client	Solved today with
Bot / crawler	IP address
Client calling an API	API keys
IoT device calling home	Proprietary attestation
Mobile app instance	Platform-specific attestation
Server-to-server	mTLS or SPIFFE or OAuth

Every answer is **platform-specific, complex, closed system, or a shared secret** — yet all of them reduce to: *sign the request, let the verifier find the key*

Call to Action

We need to solve how we authenticate clients for HTTP

- HTTP Message Signatures gave us the signature
- Signature-Key provides the missing key distribution — spanning pseudonymous → identified → delegated → PKI trust models
- Proof of possession, not shared secrets

Asks

- Is there interest in adopting this work in httpbis?
  — client authentication is bigger than bots or OAuth or SPIFFE

Questions / Discussion

- Dick Hardt dick.hardt@gmail.com
- Thibault Meunier ot-ietf@thibault.uk

- draft-hardt-httpbis-signature-key-07
- https://github.com/dickhardt/signature-key