TBD
Internet-Draft
Intended status: Standards Track
Expires: 30 October 2026

D. Hardt
Hell

28 April 2026

AAuth Bootstrap
draft-hardt-aauth-bootstrap-00

Abstract

This document defines AAuth Bootstrap, an extension to the AAuth Protocol ([I-D.hardt-aauth-protocol]) that specifies how an agent server and an agent bootstrap the agent to be bound to a person at the person's Person Server (PS). Bootstrap establishes the binding between a user (as vouched for by the user's PS) and an agent identity (aauth:local@domain) hosted by an agent server. The specification covers self-hosted agents (where the agent and agent server are co-located under a domain the user controls), web apps, and mobile agents. The agent MAY pass account hints on the PS /bootstrap request to help the PS resolve the user when the user has more than one account at the PS or when the agent already knows something about the user. Identity claims and scoped authorization are obtained separately through the standard three-party flow defined in AAuth Protocol. This specification defines the bootstrap_token, the bootstrap_endpoint on the PS and agent server, the signature schemes used at each step, and renewal flows that bypass the PS after the initial binding is established.

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/.

Copyright Notice

Table of Contents

1. Introduction

Status: Exploratory Draft

The AAuth Protocol ([I-D.hardt-aauth-protocol]) defines agent identities of the form aauth:local@domain, each bound to a signing key and published under a well-known JWKS. Bootstrap is the ceremony by which an agent server and an agent bootstrap the agent to be bound to a person at the person's PS, so that subsequent flows defined in [I-D.hardt-aauth-protocol] can run with a (user, agent) binding the PS recognizes.

The same ceremony serves several deployment shapes:

* *Self-hosted agents*, where the agent and the agent server are co-located under a domain the user controls. The agent self-issues its agent token. Binding to a PS is optional; an agent that does bind benefits from a PS that can prompt the user out of band on later auth_token requests instead of requiring an in-app interaction.
* *Web apps*, where the agent is a web application running in the user's browser and the agent server runs separately on the network. The agent server mints an agent identity for the user; the user's identity is vouched for by the user's PS, not by the agent server.
* *Mobile agents*, similar to web apps but the agent runs on the user's mobile device and uses platform attestation (App Attest, Play Integrity) to bind the agent to a hardware-protected key.

The same ceremony works whether the PS is operated by a consumer identity provider or by the user's organization (B2B). When the agent has prior context about the user — an email address from a previous session, a known organizational tenant, a corporate domain — it can pass that context as an account hint on the PS /bootstrap request; see Section 15.

Bootstrap is intentionally limited to establishing the identity binding. It does not carry scope and does not request identity claims. Identity claims (e.g., email, profile, organizational membership) are obtained separately by the agent server running the standard three-party flow defined in [I-D.hardt-aauth-protocol]. This separation keeps bootstrap focused on the binding, reuses existing protocol machinery for claims release, and lets the agent request claims incrementally rather than up front.

This document specifies that ceremony. It defines:

* The bootstrap_token, a short-lived JWT issued by the PS and consumed by the agent server.
* The /bootstrap endpoints on the PS and the agent server.
* Which signature schemes from [I-D.hardt-httpbis-signature-key] are used at each step.
* Renewal flows on the agent server that do not require re-involving the PS.
* Account hints the agent MAY pass to the PS to help resolve the user.

1.1. In Scope

* Self-hosted agents (including self-hosted CLI), where the agent and agent server are co-located under a domain the user controls. Binding to a PS is optional.
* Web apps (agent runs in the user’s browser; agent server runs separately on the network).
* Mobile agents (agent runs on the user’s device with platform attestation; agent server runs separately on the network).

B2B SaaS deployments use the same web app or mobile profile; what differs is which PS the agent talks to and which account hints it passes (see Section 15).

1.2. Out of Scope

The following deployment shapes are not specified by this document and are deferred to future revisions. Each is described briefly so implementers can recognize the pattern and understand why it is treated separately.

1.2.1. Command-line tools with a user-independent agent server

A CLI whose agent server is operated separately from the user — typically by a vendor — needs a way to enroll the CLI as an agent at the vendor without the user being present in a browser. Self-hosted CLIs are covered in Section 14; this exclusion applies only when the CLI's agent server is vendor-operated. Future work may define an out-of-band enrollment ceremony (for example, a one-time pairing code displayed on a vendor web page and entered into the CLI).

1.2.2. Server workloads and headless services

Server-side agents — daemons, batch jobs, microservices — typically run in environments that already provide a workload identity. Examples include SPIFFE SVIDs from a SPIRE agent, WIMSE workload identity tokens, AWS IMDSv2 instance identity documents, GCP metadata service tokens, and Azure IMDS tokens. The agent presents the platform attestation and an ephemeral public key to the agent server, which verifies the attestation against the platform's trust root and issues an agent token. There is no end user, so no PS interaction; the trust anchor is the platform's attestation that the workload is what it claims to be. On managed infrastructure the platform may additionally attest the software identity (container image hash, binary signature), allowing the agent server to restrict tokens to known software.

1.2.3. Managed desktops

On managed desktops (corporate MDM-enrolled devices), the management platform may provide device and software attestation comparable to mobile platforms. The agent presents the platform attestation alongside its ephemeral key, and the agent server verifies the device is managed and the software is approved. The trust anchor is the management platform's attestation rather than user interaction; consequently a different consent and enrollment model applies than for the web app or mobile profiles.

1.2.4. Organizational agents with administrative enrollment

In some enterprise deployments an organization administrator pre-authorizes agents on behalf of users — bindings exist before any user interaction. This requires a pre-established trust relationship between the organization's PS and the vendor's agent server, plus an administrative enrollment channel that this specification does not define. The user-driven case where each user bootstraps their own agent in an organizational context is covered by the in-scope profiles together with the account hints in Section 15; administrator-driven bulk enrollment is a separate problem.

2. Conventions and Definitions

{::boilerplate bcp14-tagged}

3. Terminology

3.1. Terms Reused From AAuth Protocol

The following terms are defined in [I-D.hardt-aauth-protocol] and are used in this document with the same meaning. Summaries are provided for convenience; the referenced specification is authoritative.

* *Person*: The user or organization on whose behalf an agent acts. Referred to in this document as "the user".
* *Agent*: An HTTP client acting on behalf of a person, identified by an agent identifier URI of the form aauth:local@domain defined in [I-D.hardt-aauth-protocol].
* *Agent Server*: A server that manages agent identity and issues agent tokens. In the self-hosted profile the agent server is co-located with the agent under a domain the user controls; in the web app and mobile profiles it runs separately on the network. Identified by an HTTPS URL.
* *Person Server (PS)*: A server that represents the person to the rest of the protocol. In the context of this document, the PS vouches for the user to the agent server during bootstrap. Identified by an HTTPS URL.
* *Resource*: A server that protects access to its APIs. Not involved in bootstrap.
* *Agent Token*: A JWT issued by an agent server to establish the agent’s identity, defined in [I-D.hardt-aauth-protocol].
* *Auth Token*: A JWT issued by a PS or AS granting an agent access to a resource, defined in [I-D.hardt-aauth-protocol]. Not issued during bootstrap.
* *Interaction*: User authentication or consent at an interaction endpoint, triggered when a server returns 202 Accepted with requirement=interaction. Defined in [I-D.hardt-aauth-protocol].

3.2. Terms Reused From HTTP Signature Keys

The following signature schemes are defined in [I-D.hardt-httpbis-signature-key] and used in this document without modification:

* *hwk scheme*: Conveys a public key inline in the Signature-Key header.
* *jkt-jwt scheme*: Chains a JWT signed by one key (e.g., a durable enclave key) to authorize a second key (e.g., an ephemeral HTTP signing key).

* *jwt scheme*: Conveys a JWT (e.g., an agent token) that names the signing key.

3.3. Terms Defined in This Document

* *Bootstrap*: The PS-mediated ceremony defined in this document by which an agent becomes bound to a person at that person's PS. For self-hosted agents bootstrap is optional; for web app and mobile agents it is the way to obtain an initial agent token.
* *Ephemeral Key*: An HTTP signing key generated by the agent whose lifetime is bounded by the agent token. Typically hours to a day, not per-request.
* *Durable Key*: A platform-protected key whose lifetime is bounded by the application install (or, for self-hosted agents, by the install of the agent on the user's machine). Typically months to years. Examples include the iOS Secure Enclave, Android StrongBox, the macOS Keychain (Secure Enclave on supported hardware), Windows TPM, and Linux Secret Service. A durable key is identified by a JWK thumbprint (urn:jkt).
* *bootstrap_token*: A short-lived JWT issued by the PS, directed at an agent server, and bound to an agent's ephemeral key via a cnf claim. Defined in Section 8.4.
* *Binding*: The one-to-one association (ps_url, user_sub) -> aauth:local@<agent-server-domain> recorded by the agent server at the end of bootstrap, together with the device credentials associated with it.
* *Device Credential*: A credential held by the agent server and associated with a binding, used to authenticate renewal requests without re-involving the PS. A WebAuthn credential ID for web apps or an enclave-key thumbprint (urn:jkt:sha-256:<thumbprint>) for mobile.

4. Parties and Topology

The bootstrap ceremony involves three parties:

* The *agent*, which generates key material and initiates the ceremony.
* The *PS*, which authenticates the user, collects consent, issues the bootstrap_token, and records the aauth:local@domain identifier of each agent the user bootstraps.
* The *agent server*, which verifies the bootstrap_token, performs attestation, records the (user, agent) binding, and issues the agent token.

For self-hosted agents the agent and agent server are co-located under a domain the user controls. The agent self-issues its agent token; the agent server does not expose bootstrap_endpoint or refresh_endpoint, and platform attestation is replaced by a hardware-bound JWKS-published key. See Section 14.

Resources and access servers (AS) are not involved in bootstrap. After bootstrap, the agent interacts with them using the tokens returned by the agent server following the flows defined in [I-D.hardt-aauth-protocol].

5. Bootstrap Overview

The following sequence shows the bootstrap ceremony end to end. Attestation and renewal sub-flows are shown separately for clarity.

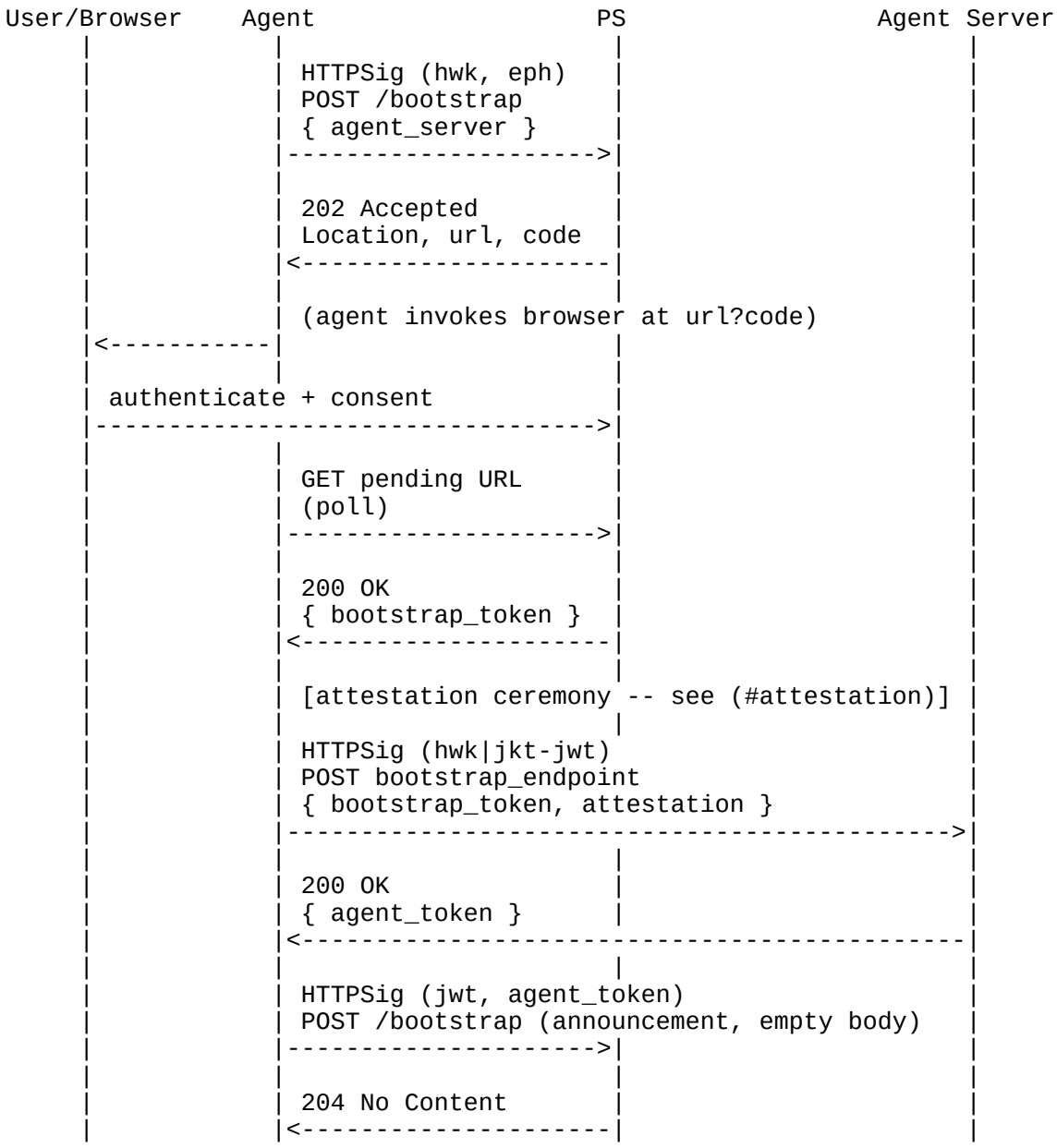

Figure 1: Bootstrap Ceremony

At this point the PS has bound the user to the agent identifier and the agent holds an agent_token.

The subsequent renewal flow skips the PS and uses the device credential recorded at bootstrap:

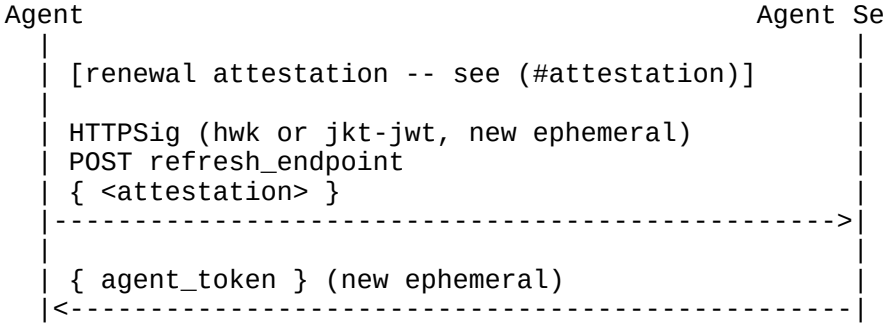

Figure 2: Renewal Ceremony

6. Agent Server Metadata Extensions

This specification extends the /.well-known/aauth-agent.json document defined in [I-D.hardt-aauth-protocol] with the following fields. These fields apply to agent servers that accept web app or mobile clients; self-hosted agent servers Section 14 do not expose these endpoints.

Fields:

* *bootstrap_endpoint* (REQUIRED for agent servers that accept web app or mobile clients). The HTTPS URL where the agent POSTs the bootstrap_token and attestation result per Section 8.5.
* *refresh_endpoint* (REQUIRED for agent servers that accept web app or mobile clients). The HTTPS URL where the agent POSTs renewal requests per Section 11.
* *webauthn_endpoint* (REQUIRED for agent servers that accept web app clients). The HTTPS URL from which the agent fetches a WebAuthn challenge and, for registration, WebAuthn ceremony options. Defined in Section 7.

Agents MUST NOT assume a fixed path for these endpoints; the agent MUST discover them from agent-server metadata. Each endpoint URL's host MUST equal the host of the issuer field, or be a DNS subdomain of it. This allows operational deployments such as https://api.agent-server.example/bootstrap paired with issuer = https://agent-server.example, while still anchoring discovery to the metadata origin.

Example aauth-agent.json file:

```
{
  "issuer": "https://agent-server.example",
  "jwks_uri": "https://agent-server.example/.well-known/jwks.json",
  "client_name": "Example AI Assistant",
  "logo_uri": "https://agent-server.example/logo.png",
  "bootstrap_endpoint": "https://agent-server.example/bootstrap",
  "refresh_endpoint": "https://agent-server.example/refresh",
  "webauthn_endpoint": "https://agent-server.example/webauthn"
}
```

7. WebAuthn Endpoint

The webauthn_endpoint issues WebAuthn challenges for both bootstrap (registration) and refresh (assertion). It is used by web app clients. The endpoint takes no user context on the request; the server tracks each challenge as an opaque single-use nonce and binds it to the user only when the ceremony result is submitted.

7.1. Request

The agent issues a GET request with the ceremony type as a query parameter:

```
GET /webauthn?type=create HTTP/1.1
Host: agent-server.example
```

or

```
GET /webauthn?type=get HTTP/1.1
Host: agent-server.example
```

Query parameters:

* *type* (REQUIRED). Either create (for bootstrap registration) or get (for refresh assertion).

The request is unsigned. The agent MUST NOT include user identifiers on the challenge request.

7.2. Response

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "challenge": "<base64url random bytes>",
  "creation_options": {
    "rp": { "id": "agent-server.example", "name": "Example AI Assistant" },
    "pubKeyCredParams": [{ "type": "public-key", "alg": -7 }, ...],
    "authenticatorSelection": { ... },
    "attestation": "none"
  }
}
```

Response members:

* *challenge* (REQUIRED). Server-generated random value of at least 16 bytes, base64url-encoded without padding. The agent server MUST store the challenge in a single-use, time-limited registry. The challenge SHOULD expire within 5 minutes.
* *creation_options* (REQUIRED when type=create). The non-user portions of a WebAuthn PublicKeyCredentialCreationOptions object ([WebAuthn]): rp, pubKeyCredParams, authenticatorSelection, attestation, timeout, and related server-decided fields. The agent fills in the challenge field from this response and fills in the user field from the bootstrap_token, deriving user.id from bootstrap_token.sub. WebAuthn ([WebAuthn], Section 5.4.3) limits user.id to 64 bytes and requires that it not contain personally identifying information; the agent MUST encode bootstrap_token.sub accordingly (typically the raw 32-byte SHA-256 of sub, since sub is already a pairwise opaque identifier). MUST be absent when type=get.

The response MUST NOT include user-identifying information. The agent server MUST NOT bind the challenge to a particular user in its issuing state; user context enters only when the ceremony result is submitted.

8. Bootstrap Flow

The bootstrap flow consists of the following steps.

8.1. Key Generation

The agent generates an ephemeral signing key.

On mobile platforms, the agent also has (or generates on first install) a durable enclave-protected key with a stable JWK thumbprint of the form urn:jkt:sha-256:<thumbprint>.

In the web app profile, only the ephemeral key exists. (The self-hosted profile uses a JWKS-published hardware-bound key in place of these; see Section 14.)

8.2. Request to PS /bootstrap

The agent sends an HTTP request to the PS's /bootstrap endpoint signed under the hwk scheme ([I-D.hardt-httpbis-signature-key]) using the ephemeral key:

```
POST /bootstrap HTTP/1.1
Host: ps.example
Signature-Input: sig=...
Signature: sig=...
Signature-Key: sig=hwk;kty="OKP";crv="Ed25519";x="<ephemeral-pubkey>"
Content-Type: application/json

{
  "agent_server": "https://agent-server.example"
}
```

The request body is a JSON object with the following members:

* *agent_server* (REQUIRED). The HTTPS URL identifying the agent server that will mint the agent identity. This value is placed in the aud claim of the issued bootstrap_token.
* *tenant*, *domain_hint*, *login_hint* (OPTIONAL account hints). See Section 15 for semantics and selection guidance.

8.3. Interaction Response

The PS responds with an interaction requirement using the requirement-response and polling pattern defined in [I-D.hardt-aauth-protocol]:

```
HTTP/1.1 202 Accepted
Location: https://ps.example/bootstrap/pending/abc123
Retry-After: 0
Cache-Control: no-store
AAuth-Requirement: requirement=interaction;
    url="https://ps.example/interaction"; code="A1B2-C3D4"
Content-Type: application/json

{
  "status": "pending"
}
```

The agent constructs the user-facing URL from the url and code parameters and directs the user to it per the methods defined in [I-D.hardt-aauth-protocol] (browser redirect, QR code, or display code). The PS authenticates the user and presents a consent screen asking the user to allow the agent server to establish an account bound to the user at this PS. No identity claims are released at this step; claims flow through the standard three-party flow defined in [I-D.hardt-aauth-protocol] after bootstrap completes.

The consent screen SHOULD display the agent server's display name and logo as retrieved from the agent server's /.well-known/aauth-agent.json metadata document, alongside the agent server's host. The user approves or denies the request.

The agent polls the Location URL until the interaction completes, following the polling rules in [I-D.hardt-aauth-protocol]. While the interaction is pending the PS returns the same 202 envelope; on completion it returns the response defined in Section 8.4.

8.4. bootstrap_token Issuance

On successful user approval, a poll of the pending URL returns the bootstrap_token:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "bootstrap_token": "eyJhbGc..."
}
```

The bootstrap_token is a signed JWT [RFC7519] with the following structure.

8.4.1. bootstrap_token Header

* *alg*: The signature algorithm (e.g., EdDSA).
* *typ*: aa-bootstrap+jwt.
* *kid*: Key identifier for the PS key used to sign the token.

8.4.2. bootstrap_token Payload

* *iss* (REQUIRED). The PS URL.
* *dwk* (REQUIRED). The well-known document name used to discover the PS's JWKS, defined in [I-D.hardt-httpbis-signature-key]. For a PS this is aauth-person.json.
* *aud* (REQUIRED). The agent server URL (matches agent_server from the request).

* *sub* (REQUIRED). A pairwise user identifier, directed at aud, identifying the user to the agent server.
* *cnf* (REQUIRED). Confirmation claim ([RFC7800]) with jwk containing the agent's ephemeral public key. The key MUST match the hwk key used to sign the PS /bootstrap request.
* *jti* (REQUIRED). A unique token identifier per [RFC7519].
* *iat* (REQUIRED). Issued-at time per [RFC7519].
* *exp* (REQUIRED). Expiration time per [RFC7519]. SHOULD NOT exceed 5 minutes after iat.

The bootstrap_token does not carry scope, agent, or any claim describing user attributes. Its sole purpose is to convey a directed user identifier (sub) bound to an agent-side ephemeral key (cnf). Identity claims are obtained separately via the three-party flow defined in [I-D.hardt-aauth-protocol] after bootstrap.

The bootstrap_token differs from the resource token defined in [I-D.hardt-aauth-protocol]:

* It omits the agent claim, because the agent identity does not yet exist at bootstrap time. The binding to the agent is carried solely by cnf.
* It is directed at an agent server, not at a PS or AS.

8.5. Request to Agent Server /bootstrap

This step is omitted for self-hosted agents Section 14, where the agent and agent server are co-located and the agent self-issues the agent token instead of POSTing to a remote bootstrap_endpoint. The remainder of this section applies to web app and mobile profiles.

Before calling bootstrap_endpoint, the agent performs the attestation ceremony appropriate to its platform. See Section 9 for the ceremony details and for the logic by which the platform is chosen. The attestation result is carried in the POST body.

The agent sends a single POST to bootstrap_endpoint:

```
POST /bootstrap HTTP/1.1
Host: agent-server.example
Signature-Input: sig=...
Signature: sig=...
Signature-Key: sig=<scheme>
Content-Type: application/json

{
  "bootstrap_token": "<jwt>",
  "attestation": {
    "type": "<attestation type>",
    ...
  }
}
```

The HTTP Message Signature scheme is determined by the platform:

* *Web app clients* use hwk with the agent's ephemeral key.
* *Mobile clients* use jkt-jwt ([I-D.hardt-httpbis-signature-key]), where the JWT header references the enclave key and the JWT payload's cnf.jwk is the ephemeral public key. The ephemeral key signs the HTTP message.

Request body members:

* *bootstrap_token* (REQUIRED). The JWT received from the PS.
* *attestation* (REQUIRED). The attestation result, whose shape is defined by the ceremony described in Section 9. The type field identifies the ceremony (e.g., webauthn, app-attest, play-integrity) and MUST match the scheme implied by the HTTP signature.

On receiving the request, the agent server MUST:

1. Verify the HTTP Message Signature per [RFC9421] and [I-D.hardt-httpbis-signature-key].
2. Resolve and validate the bootstrap_token signature per [RFC7515] and [RFC7519] by fetching the PS's JWKS using iss and dwk.
3. Apply local policy on whether bootstrap_token.iss is an acceptable PS for this agent server. By default, consumer-profile agent servers SHOULD accept any PS that publishes valid metadata; B2B-profile agent servers SHOULD restrict acceptance to an explicit allowlist. Reject with 403 Forbidden if policy denies.
4. Verify that bootstrap_token.aud equals the agent server's own URL.
5. Verify that bootstrap_token.cnf.jwk matches the ephemeral public key used to sign the HTTP request.

6. Verify iat is not in the future and exp is in the future per [RFC7519]; reject replays by jti.
7. Verify the attestation per the rules of the indicated ceremony in Section 9, including that any ceremony challenge or nonce matches one issued by the agent server and has not expired or been consumed. Mark the ceremony challenge as consumed.
8. Create or look up the binding Section 8.6.
9. Store the device credential derived from the attestation (a WebAuthn credential ID for web apps, or the enclave key thumbprint for mobile) on the binding.
10. Issue the agent token Section 8.7.

If bootstrap_token verification fails, the agent server MUST respond 400 Bad Request. If issuer policy denies, the agent server MUST respond 403 Forbidden. If attestation verification fails, the agent server MUST respond 401 Unauthorized.

8.6. Binding Creation

The agent server looks up or creates a binding keyed by (bootstrap_token.iss, bootstrap_token.sub):

(ps_url, user_sub) -> aauth:local@<agent-server-domain>

The binding is one-to-one: the same user at the same PS at the same agent server MUST map to the same aauth:local@domain identity, regardless of device.

The agent server stores the device credential obtained during attestation (a WebAuthn credential ID, or a urn:jkt:sha-256:<enclave> thumbprint) against the binding. Multiple device credentials MAY be associated with a single binding.

8.7. Token Response

The agent server returns an agent token:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "agent_token": "eyJhbGc..."
}
```

The agent_token is a JWT [RFC7519] as defined in [I-D.hardt-aauth-protocol] with:

* *typ*: aa-agent+jwt.

* *iss*: The agent server URL.
* *dwk*: aauth-agent.json.
* *sub*: The aauth:local@<agent-server-host> identity from the binding.
* *ps*: The PS URL (from bootstrap_token.iss).
* *cnf.jwk*: The agent's ephemeral public key.

The agent server does not issue a resource token at bootstrap. If the agent server needs identity claims about the user (e.g., to populate a profile page), it follows the standard three-party flow defined in [I-D.hardt-aauth-protocol] after bootstrap completes.

8.8. Bootstrap Completion

Once the agent holds the agent_token, it SHOULD announce its new agent identity to the PS so the PS can bind the identity to the user within the PS's bootstrap record. The agent SHOULD perform this announcement immediately upon receiving the agent_token and in any case before bootstrap_token.exp, because the PS correlates the announcement to the bootstrap record by the ephemeral key's thumbprint and MAY discard records after expiry. The agent MUST perform the announcement before rotating its ephemeral key (that is, before any call to refresh_endpoint).

8.8.1. Announcement Request

The agent sends an empty POST to the PS's /bootstrap endpoint, signed under the jwt scheme ([I-D.hardt-httpbis-signature-key]) with the agent_token as the naming JWT:

```
POST /bootstrap HTTP/1.1
Host: ps.example
Signature-Input: sig=...
Signature: sig=...
Signature-Key: sig=jwt;jwt="<agent_token>"
Content-Length: 0
```

The PS distinguishes the announcement from the initial bootstrap by the signature scheme and the empty body: the initial call uses hwk with a JSON body, the announcement uses jwt with an empty body.

8.8.2. PS Processing

On receiving the announcement, the PS MUST:

1. Verify the HTTP Message Signature per [RFC9421] and [I-D.hardt-httpbis-signature-key] under the jwt scheme.

2. Verify the agent_token per [RFC7515] and [RFC7519] by resolving the agent server's JWKS via agent_token.iss and agent_token.dwk per [I-D.hardt-httpbis-signature-key].
3. Verify that agent_token.ps equals this PS's URL.
4. Look up the bootstrap record by the thumbprint of agent_token.cnf.jwk.
5. If a matching bootstrap record exists, record the binding between agent_token.sub (the aauth:local@domain identifier) and the (user, agent_server) tuple already on file, then respond 204 No Content.
6. If no matching record exists, respond 404 Not Found.

The announcement is idempotent: repeated calls for the same ephemeral thumbprint after a successful binding have no effect and respond 204 No Content.

The PS MUST retain the bootstrap record at least until bootstrap_token.exp, plus a small allowance for clock skew (a few seconds is sufficient). After that time the PS MAY discard the record; an announcement that arrives later MAY fail with 404. Even when the explicit announcement fails, the PS MAY learn the binding lazily from the agent claim of a resource token presented at the PS /token endpoint during the standard three-party flow.

8.8.3. Post-Bootstrap API Access

With the agent_token in hand and the announcement made, the agent uses the agent_token to call the agent server's APIs:

* *Identity-based calls*, where the agent server authorizes based solely on the agent's identity, use the agent_token directly per the identity-based mode of [I-D.hardt-aauth-protocol].
* *Calls requiring user claims* follow the standard three-party flow defined in [I-D.hardt-aauth-protocol]: the agent calls the agent server's authorization_endpoint to obtain a resource token, exchanges that resource token at the PS /token endpoint for an auth token carrying the required claims, then calls the agent server's API with the auth token.

Each claim-bearing call follows the same pattern, which lets the agent request claims incrementally rather than up front. The PS applies its own policy (including any user consent) when issuing each auth token.

9. Attestation

Every call in this specification that mints tokens for an agent (at bootstrap or at renewal) carries a platform-specific attestation, except for self-hosted agents Section 14 where the trust anchor is the user-controlled JWKS-published key. At bootstrap the attestation registers a device credential on the binding; at renewal it proves possession of that device credential. Without attestation an attacker who intercepted a bootstrap_token or replayed a refresh could obtain tokens from a machine the user does not control.

9.1. Ceremony Selection

The agent selects the ceremony from its runtime platform. The agent server determines which ceremony to expect from the HTTP signature scheme on the bootstrap_endpoint or refresh_endpoint POST.

Platform	Bootstrap ceremony	Renewal ceremony	HTTP signature scheme
Self-hosted	None (hardware-bound JWKS key)	None (hardware-bound JWKS key)	jwt (self-issued agent token)
Web app (browser-hosted)	WebAuthn registration ([WebAuthn])	WebAuthn assertion ([WebAuthn])	hwk
Mobile (iOS)	App Attest	Enclave jkt-jwt	jkt-jwt
Mobile (Android)	Play Integrity	Enclave jkt-jwt	jkt-jwt

Table 1

An agent server that accepts web app clients MUST publish webauthn_endpoint in its metadata Section 6.

For mobile clients, the agent server MUST nominate the nonce consumed by the platform attestation; the nonce MUST be at least 16 bytes of cryptographically random data, single-use, and MUST expire within 5 minutes of issuance. The transport by which the agent obtains the nonce is application-defined: most mobile agents already share an authenticated channel with their server and can carry the nonce on an existing endpoint. No metadata field for this transport is defined here. Section 9.3.3 gives a non-normative example shape for implementations that do not have an existing channel.

Self-hosted agents do not use bootstrap_endpoint or refresh_endpoint and therefore do not perform a ceremony at the agent server step; the trust anchor is the hardware-bound JWKS key described in Section 14.

9.2. Web App: WebAuthn

Web app clients use the same webauthn_endpoint Section 7 for both bootstrap (type=create) and renewal (type=get). The ceremony result is carried in the POST body to bootstrap_endpoint or refresh_endpoint.

9.2.1. Registration (at Bootstrap)

The agent first fetches a challenge and creation options from webauthn_endpoint:

```
GET /webauthn?type=create HTTP/1.1
Host: agent-server.example
```

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "challenge": "<base64url random bytes>",
  "creation_options": {
    "rp": { "id": "agent-server.example", "name": "Example AI Assistant" },
    "pubKeyCredParams": [
      { "type": "public-key", "alg": -7 },
      { "type": "public-key", "alg": -257 }
    ],
    "authenticatorSelection": {
      "residentKey": "required",
      "userVerification": "preferred"
    },
    "attestation": "none"
  }
}
```

The agent invokes navigator.credentials.create() using those options, filling user.id with bootstrap_token.sub and supplying user.name and user.displayName from agent-side data. The resulting PublicKeyCredential is included in the POST to bootstrap_endpoint:

```
POST /bootstrap HTTP/1.1
Host: agent-server.example
Signature-Input: sig=...
Signature: sig=...
Signature-Key: sig=hwk;kty="OKP";crv="Ed25519";x="<ephemeral-pubkey>"
Content-Type: application/json

{
  "bootstrap_token": "<jwt>",
  "attestation": {
    "type": "webauthn",
    "credential": {
      "id": "<base64url credential id>",
      "rawId": "<base64url>",
      "response": {
        "clientDataJSON": "<base64url>",
        "attestationObject": "<base64url>"
      },
      "type": "public-key"
    }
  }
}
```

The agent server verifies the WebAuthn registration ([WebAuthn]) including that clientDataJSON.challenge matches a challenge it issued via webauthn_endpoint, and records the credential ID as a device credential on the binding.

9.2.2. Assertion (at Renewal)

The agent first fetches a challenge from webauthn_endpoint:

```
GET /webauthn?type=get HTTP/1.1
Host: agent-server.example

HTTP/1.1 200 OK
Content-Type: application/json

{
  "challenge": "<base64url random bytes>"
}
```

The agent invokes navigator.credentials.get() with that challenge and the agent server's rpId, using a discoverable credential (no allowCredentials). The resulting PublicKeyCredential is included in the POST to refresh_endpoint:

```
POST /refresh HTTP/1.1
Host: agent-server.example
Signature-Input: sig=...
Signature: sig=...
Signature-Key: sig=hwk;kty="OKP";crv="Ed25519";x="<new-ephemeral-pubkey>"
Content-Type: application/json

{
  "attestation": {
    "type": "webauthn",
    "credential": {
      "id": "<base64url credential id>",
      "rawId": "<base64url>",
      "response": {
        "clientDataJSON": "<base64url>",
        "authenticatorData": "<base64url>",
        "signature": "<base64url>",
        "userHandle": "<base64url>"
      },
      "type": "public-key"
    }
  }
}
```

The agent server looks up the binding by the credential's rawId and verifies the WebAuthn assertion ([WebAuthn]) including that clientDataJSON.challenge matches a challenge it issued via webauthn_endpoint.

9.3. Mobile: Platform Attestation and Enclave Proof

Mobile clients use App Attest ([AppAttest]) on iOS or Play Integrity ([PlayIntegrity]) on Android at bootstrap to enroll the enclave key as the device credential. At renewal, the enclave key signature in the jkt-jwt HTTP Message Signature is itself sufficient proof; no additional ceremony is required.

9.3.1. Initial Attestation (at Bootstrap)

The agent obtains a nonce from the agent server (subject to the constraints in Section 9.1) and feeds it to the platform attestation API. On iOS the nonce is bound by computing clientDataHash over the nonce; on Android the nonce is supplied to the Play Integrity request and surfaced in requestDetails.nonce of the verdict.

Once the ceremony has produced an attestation, the agent POSTs to bootstrap_endpoint:

```
POST /bootstrap HTTP/1.1
Host: agent-server.example
Signature-Input: sig=...
Signature: sig=...
Signature-Key: sig=jkt-jwt;jwt="<jkt-jwt-token>"
Content-Type: application/json

{
  "bootstrap_token": "<jwt>",
  "attestation": {
    "type": "app-attest",
    "key_id": "<base64url>",
    "attestation_object": "<base64url>",
    "client_data_hash": "<base64url>"
  }
}
```

For Play Integrity the attestation object has the form:

```
{
  "type": "play-integrity",
  "integrity_token": "<JWS-encoded attestation>"
}
```

The agent server verifies the attestation per the platform's published rules, including that the nonce was one it nominated, and records the enclave key thumbprint (urn:jkt:sha-256:<thumbprint>) as the device credential on the binding.

9.3.2. Enclave Proof (at Renewal)

The agent generates a new ephemeral key, has the enclave sign a jkt-jwt binding that key, and POSTs to refresh_endpoint:

```
POST /refresh HTTP/1.1
Host: agent-server.example
Signature-Input: sig=...
Signature: sig=...
Signature-Key: sig=jkt-jwt;jwt="<jkt-jwt-token>"
```

The request body is empty. The agent server computes the enclave key thumbprint from the jkt-jwt header, looks up the binding, and issues a fresh agent_token. No attestation member appears in the body because the jkt-jwt signature itself is the proof.

9.3.3. Non-Normative Nonce Endpoint Example

This non-normative example shows one shape an agent server might expose if it does not already have an authenticated channel suitable for delivering the attestation nonce. It mirrors the webauthn_endpoint pattern: an unsigned GET that returns an opaque single-use nonce. Implementations are free to use any other transport that meets the constraints in Section 9.1.

Request:

```
GET /attestation/nonce HTTP/1.1
Host: agent-server.example
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json
Cache-Control: no-store

{
  "nonce": "<base64url random bytes>"
}
```

The agent feeds nonce to the platform attestation API per Section 9.3.1.

10. Signature Scheme Summary

The following signature schemes from [I-D.hardt-httpbis-signature-key] are used at each step of bootstrap and renewal:

Context	Scheme	Key material
PS /bootstrap initial request	hwk	Ephemeral (inline)
PS /bootstrap announcement request	jwt	Ephemeral (via agent_token)
Agent Server webauthn_endpoint request (web app)	unsigned	(no signature)
Agent Server bootstrap_endpoint request (web app)	hwk	Ephemeral (same as PS call)
Agent Server bootstrap_endpoint request (mobile)	jkt-jwt	Enclave signs ephemeral
Agent Server refresh_endpoint request (web app)	hwk + WebAuthn assertion in body	New ephemeral + user proof
Agent Server refresh_endpoint request (mobile)	jkt-jwt	Enclave signs new ephemeral
Self-hosted: agent self-issues agent_token	n/a (no agent server endpoint call)	JWKS-published hardware-bound key
Post-bootstrap resource calls	jwt	agent_token wrapping ephemeral

Table 2

11. Renewal

Agent tokens expire. Per [I-D.hardt-aauth-protocol] the maximum lifetime is 24 hours. Renewal bypasses the PS because the agent server already holds the (user, agent) binding and the device credential recorded at bootstrap.

Agent servers that accept web app or mobile clients MUST expose a refresh_endpoint as defined in Section 6. The endpoint issues a fresh agent_token bound to a new ephemeral key.

11.1. Renewal Flow

Before calling refresh_endpoint, the agent performs the renewal attestation ceremony appropriate to its platform. See Section 9 for the ceremony details. The ceremony result, if any, is carried in the POST body.

```
POST /refresh HTTP/1.1
Host: agent-server.example
Signature-Input: sig=...
Signature: sig=...
Signature-Key: sig=<scheme>
Content-Type: application/json

{
  "attestation": {
    "type": "<attestation type>",
    ...
  }
}
```

The HTTP Message Signature scheme is determined by the platform:

* *Web app clients* use hwk with a newly generated ephemeral key.
* *Mobile clients* use jkt-jwt ([I-D.hardt-httpbis-signature-key]), where the enclave key signs a jkt-jwt binding a newly generated ephemeral key; the ephemeral key signs the HTTP message. For mobile, the request body MAY be empty because the jkt-jwt signature itself proves possession of the device credential.

Request body members:

* *attestation* (REQUIRED for web app clients; omitted for mobile clients). The assertion result defined by the renewal ceremony in Section 9.

On receiving the request, the agent server MUST:

1. Verify the HTTP Message Signature per [RFC9421] and [I-D.hardt-httpbis-signature-key].
2. Look up the binding:
    * For web app clients, by the credential ID (rawId) in the WebAuthn assertion.
    * For mobile clients, by the enclave key thumbprint (urn:jkt:sha-256:<thumbprint>) derived from the jkt-jwt header.

3. Verify the renewal attestation per the rules of the indicated ceremony in Section 9, including that any ceremony challenge matches one it issued and has not expired or been consumed. Mark the challenge as consumed.
4. Issue a fresh agent token bound to the new ephemeral key.

11.2. Refresh Response

On success, the agent server returns:

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "agent_token": "eyJhbGc..."
}
```

The agent_token has the same structure as in Section 8.7 but bound to the new ephemeral key.

11.3. Renewal Failure

Renewal MUST fail in the following conditions. The agent MUST then repeat the full PS-mediated bootstrap flow from Section 8. The agent server distinguishes the failure mode by status code:

* *404 Not Found* when no binding matches the credential identifier presented (e.g., the enclave key has been regenerated after mobile reinstall and the urn:jkt is unknown; the WebAuthn rawId is unknown; or the binding has been revoked at the agent server and removed from storage).
* *401 Unauthorized* when a binding is found but the proof fails (e.g., a WebAuthn assertion does not verify, the jkt-jwt signature does not verify, or the ceremony challenge has expired or been consumed).

Independently, the PS MAY have stopped vouching for the user (for example, account closure or organizational deprovisioning). The agent server has no direct signal of this; it surfaces only through a subsequent PS /bootstrap call that fails at the interaction step. The agent will then be unable to re-establish the binding.

12. Agent-Person Binding Invariant

Per [I-D.hardt-aauth-protocol], each agent is bound to exactly one person. In bootstrap:

* (ps_url, user_sub) -> aauth:local@<agent-server-domain> is one-to-one.
* The same user at the same PS maps to the same aauth:local@<agent-server-domain> regardless of device.
* Multiple devices per binding MAY exist through multiple associated device credentials.
* Revoking a binding at the agent server revokes access for all devices bound to it.

This supports:

* Account switching, where an agent holds multiple agent tokens for multiple (user, agent) bindings and selects among them at runtime.
* Multi-device use, where a single aauth:local identity has device credentials on several devices.
* Immediate cross-device revocation, where disabling the binding at the agent server invalidates all device credentials simultaneously.

13. Out-of-Band User Channel

After first bootstrap, the PS knows the (user, agent_server) binding and MAY establish a communication channel with the user (push, email, or an authenticated session). PSes that establish such a channel at first bootstrap can use it to prompt the user for consent on subsequent auth_token requests from the agent server without requiring an in-app interaction.

This does not change the bootstrap protocol itself; bootstrap produces only the binding. It enables a smoother user experience for the standard three-party flow by letting the PS reach the user out of band when a token request requires consent.

PS implementations that wish to support out-of-band consent SHOULD establish a direct user communication channel at first bootstrap.

14. Self-Hosted Agents

In a self-hosted deployment the agent and agent server are co-located under a domain the user controls. The user publishes the agent server's /.well-known/aauth-agent.json metadata document and JWKS at that domain per [I-D.hardt-aauth-protocol]. The corresponding private signing key is held on the user's machine. Self-issued agent tokens are signed by that key and verified by any party against the published JWKS.

JWKS publication, key handling, and key rotation for the agent server role are unchanged from [I-D.hardt-aauth-protocol]. The JWKS-published key used to sign agent tokens SHOULD be hardware-bound where the platform supports it: the macOS Keychain (Secure Enclave on supported hardware), Windows TPM, or Linux Secret Service. This mirrors the durable-key guidance for mobile enclaves elsewhere in this document and is the trust anchor that takes the place of platform attestation in the self-hosted profile.

A self-hosted agent operates in one of two modes:

14.1. Without a PS

The agent self-issues an agent_token with no ps claim. The agent does not call the PS /bootstrap endpoint and does not call the agent server /bootstrap endpoint (it would be calling itself). The agent uses its self-issued agent_token directly with resources that support identity-based or resource-managed access per [I-D.hardt-aauth-protocol]. Resources requiring PS-issued auth tokens are not reachable in this mode.

14.2. With a PS

The agent calls the PS /bootstrap endpoint to establish a (user, agent) binding at the PS, then self-issues an agent_token carrying a ps claim that names the bound PS. The PS-side flow is unchanged from Section 8:

1. The agent generates an ephemeral key and POSTs to the PS /bootstrap endpoint signed under the hwk scheme, with agent_server set to the agent server's URL (the user's domain rooted at https://).
2. The PS authenticates the user, collects consent, and returns a bootstrap_token with aud set to the agent server's URL and sub set to a directed user identifier per Section 8.4.
3. The agent does not POST to its own bootstrap_endpoint. Instead, the agent self-issues an agent_token per Section 8.7 with iss set to the agent server's URL, cnf.jwk set to the same ephemeral key bound in bootstrap_token.cnf.jwk, and ps set to the PS URL from bootstrap_token.iss. The aauth:local@<agent-server-host> sub of the agent_token is chosen by the agent under its own host.
4. The agent performs the bootstrap completion announcement to the PS per Section 8.8. The PS verifies the self-issued agent_token by resolving the agent server's JWKS at agent_token.iss, exactly as it would for an agent server operated separately from the agent.

After the announcement, the PS holds the (user, agent) binding. When a resource later requires a PS-issued auth token and the PS needs the user's consent, the PS MAY prompt the user out of band over a channel established at first bootstrap Section 13 instead of requiring an in-app interaction.

15. Account Hints

When the agent has prior context about who the user is, it MAY pass an account hint on the PS /bootstrap request to help the PS resolve the user without prompting them to choose:

* *login_hint* ([OpenID.Core], Section 3.1.2.1) — an identifier for the user, typically an email address. Useful in both consumer (B2C) and organizational (B2B) contexts when the agent has learned the user's identifier from a previous session, a sign-up form, or another out-of-band source.
* *domain_hint* ([OpenID.Enterprise]) — a DNS domain associated with the user. Typically used in B2B contexts where the agent knows the user's company domain but not the specific user.
* *tenant* ([OpenID.Enterprise]) — an organizational tenant identifier known to the PS. Used in B2B contexts where the agent has been invoked in an organization-specific context (e.g., from a workspace launcher).

The PS handles these hints with the same semantics an OpenID Provider applies to the corresponding OpenID Connect parameters: a hint is advisory and the PS MAY override or ignore it based on local policy. The agent SHOULD send only the most specific hint it has.

A request might look like:

```
{
  "agent_server": "https://vendor.example",
  "login_hint": "user@example.com"
}
```

When the user has more than one account at the PS (e.g., a personal account and a work account), the hint tells the PS which to select. The selected account affects which pairwise sub the PS issues and which account subsequent token requests resolve against. When no hint is sent and more than one account is available, the PS MAY prompt the user to choose.

The bootstrap_token carries no identity or organization-related claims; those are released in the auth token issued by the PS on the standard three-party flow per [I-D.hardt-aauth-protocol]. This lets the agent server apply user- or organization-based authorization without running a per-customer SAML or OIDC integration.

16. Security Considerations

16.1. bootstrap_token Replay

The bootstrap_token carries a cnf claim binding it to the agent's ephemeral public key. Possession of the token alone is insufficient; the holder MUST also control the corresponding private key. Agent servers MUST reject bootstrap_tokens that are not accompanied by an HTTP request signed with the key named in cnf.jwk, and MUST reject replay by jti.

16.2. Consent Phishing

The user sees the agent server's domain, name, and logo at the PS consent screen. This relies on user recognition of the agent server. The PS SHOULD retrieve the agent server's display metadata from its /.well-known/aauth-agent.json document and present it at the consent screen.

16.3. Key-to-User Binding Without Attestation

For the web app and mobile profiles, the bootstrap flow relies on platform attestation at the agent server step. Environments without a platform attestation mechanism cannot safely use this flow because a remote process may impersonate a local user by relaying ephemeral keys through them. This is the reason vendor-operated command-line tools are out of scope; self-hosted agents Section 14 avoid the problem because the trust anchor is a hardware-bound JWKS-published key controlled by the user.

16.4. Enclave Key Compromise

On mobile, compromise of the enclave key breaks the chain of delegated ephemeral keys. Implementations SHOULD use the shortest practical jkt-jwt lifetimes.

16.5. Agent Server Compromise

Compromise of an agent server breaks all agent identities minted by that server. Bootstrap does not introduce this risk, but centralizes it at the agent server.

16.6. PS Compromise

The PS is already a high-value target in [I-D.hardt-aauth-protocol]. Bootstrap does not change the risk profile, but makes the PS load-bearing for agent identity creation.

16.7. Bootstrap Completion Announcement

The announcement POST to the PS Section 8.8 is bound to possession of the agent's ephemeral key plus a signed agent_token issued to that key. An attacker would need both the ephemeral private key and a valid agent_token for the target aauth identifier. These are the same credentials that protect the rest of bootstrap; the announcement introduces no new attack surface.

16.8. Resource Exhaustion at Unauthenticated Endpoints

Three endpoints accept unauthenticated requests and allocate server-side state:

* The PS /bootstrap endpoint creates a pending bootstrap record and an interaction code.
* The agent server webauthn_endpoint issues a single-use challenge.
* The non-normative mobile nonce endpoint of Section 9.3.3, if exposed, issues a single-use nonce.

An attacker can flood any of these to exhaust storage or CPU. Implementations MUST rate-limit these endpoints (per source IP and globally), keep the issued state small (an opaque random value plus expiry), and apply short TTLs (5 minutes is RECOMMENDED) so that orphaned state is reclaimed quickly. Implementations SHOULD also apply rate limiting to bootstrap_endpoint and refresh_endpoint even though those are signed, since signature verification is itself non-trivial work.

17. Privacy Considerations

17.1. Directed User Identifiers

The bootstrap_token.sub claim MUST be a pairwise user identifier directed at the agent server. This prevents cross-vendor correlation of users across different agent servers.

17.2. Mobile Enclave Identity

The PS sees only the ephemeral hwk key. The agent server sees the jkt-jwt carrying the enclave identity. The PS therefore cannot track a device across bootstraps at different agent servers.

17.3. Out-of-Band Consent Channel

The user communication channel held by the PS after first bootstrap is privacy-sensitive. PS implementations SHOULD document their user communication practices.

17.4. Agent Identifier Registry at the PS

After bootstrap completion Section 8.8, the PS knows the aauth:local@domain identifier of each agent the user has bootstrapped. This supports user-facing features such as a dashboard of connected agents and targeted revocation ("disconnect from agent X"). PS implementations SHOULD make this list visible to the user.

17.5. Announcement Metadata Exposure

The bootstrap completion announcement Section 8.8 is sent over TLS, but its fact-of-existence and timing reveal the (PS, agent_server, agent_id) triple to any on-path observer that can correlate the connection’s destination with the agent_token carried in Signature-Key. Operators of either side SHOULD treat the announcement timing with the same care as any other consent-related event, and SHOULD avoid logging the agent_token value in shared infrastructure.

18. IANA Considerations

18.1. Media Type Registrations

This specification registers the following media type in the IANA Media Types registry:

18.1.1. application/aa-bootstrap+jwt

* Type name: application
* Subtype name: aa-bootstrap+jwt
* Required parameters: N/A
* Optional parameters: N/A
* Encoding considerations: binary; a JWT is a sequence of Base64url-encoded parts separated by period characters
* Security considerations: See Section 16
* Interoperability considerations: N/A
* Published specification: This document, Section 8.4
* Applications that use this media type: AAuth Person Servers and agent servers
* Fragment identifier considerations: N/A

18.2. JWT Type Registrations

This specification registers the following JWT typ header parameter value in the "JSON Web Token Types" sub-registry:

Type Value	Reference
aa-bootstrap+jwt	This document, Section 8.4

Table 3

18.3. JWT Claims

This document defines no new JWT claims. The bootstrap_token uses claims registered in the IANA "JSON Web Token Claims" registry ([RFC7519], [RFC7800]) together with dwk, which is registered by [I-D.hardt-aauth-protocol].

18.4. AAuth Agent Server Metadata Registration

This document registers the following parameters in the AAuth Agent Server Metadata registry established by [I-D.hardt-aauth-protocol]:

Parameter	Description	Reference
bootstrap_endpoint	URL of the agent server's bootstrap endpoint	This document, Section 6
refresh_endpoint	URL of the agent server's refresh endpoint	This document, Section 6
webauthn_endpoint	URL of the agent server's WebAuthn challenge endpoint	This document, Section 7

Table 4

19. 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.

There are currently no known implementations.

20. Document History

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

* draft-hardt-aauth-bootstrap-00
  - Initial submission.

21. Acknowledgments

TBD.

22. References

22.1. Normative References

[I-D.hardt-aauth-protocol]
Hardt, D., "AAuth Protocol", 2026, <https://github.com/dickhardt/AAuth>.

[I-D.hardt-httpbis-signature-key]
Hardt, D., "HTTP Signature Keys", 2026, <https://datatracker.ietf.org/doc/draft-hardt-httpbis-signature-key>.

[OpenID.Core]
Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., and C. Mortimore, "OpenID Connect Core 1.0", November 2014, <https://openid.net/specs/openid-connect-core-1_0.html>.

[OpenID.Enterprise]
Hardt, D. and K. McGuinness, "OpenID Connect Enterprise Extensions 1.0", 2025, <https://openid.net/specs/openid-connect-enterprise-extensions-1_0.html>.

[RFC7515] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", RFC 7515, DOI 10.17487/RFC7515, May 2015, <https://www.rfc-editor.org/info/rfc7515>.

[RFC7519] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", RFC 7519, DOI 10.17487/RFC7519, May 2015, <https://www.rfc-editor.org/info/rfc7519>.

[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>.

[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>.

22.2. Informative References

[AppAttest] Apple, "Establishing your app’s integrity (App Attest)", 2024, <https://developer.apple.com/documentation/devicecheck/establishing-your-app-s-integrity>.

[PlayIntegrity] Google, "Play Integrity API", 2024, <https://developer.android.com/google/play/integrity/overview>.

[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>.

[WebAuthn] W3C, "Web Authentication: An API for accessing Public Key Credentials - Level 3", 2024, <https://www.w3.org/TR/webauthn-3/>.

Author’s Address

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