Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: 18 November 2026

S. Jovancevic
SKGO, IKT Support

18 May 2026

Browser Vendor Attestation Protocol (BVAP)
draft-jovancevic-bvap-01

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 18 November 2026.

Copyright Notice

Abstract

The HTTP User-Agent header field allows any client to claim any browser identity without cryptographic proof. Any software may assert "Mozilla/5.0 Chrome/124" regardless of its actual origin, distribution source, or software provenance.

This document specifies BVAP (Browser Vendor Attestation Protocol), a mechanism by which browser vendors cryptographically attest that a browser instance originates from a legitimate, vendor-issued browser build. Attestation occurs at installation time and is periodically renewed. The resulting Vendor Attestation Seal (BVAP Seal) is carried by the browser and is primarily transmitted via the Sec-BVAP header field. The Seal MAY additionally appear in the User-Agent string for backward compatibility with legacy infrastructure.

BVAP provides software provenance and vendor authenticity at the browser layer. The protocol allows servers to distinguish between:

- Vendor-attested browser software,
- Anonymous or self-built browser software (Unattested), and
- Software falsely claiming vendor browser identity.

BVAP requires no per-request cryptography and no DNS lookups during normal operation.

BVAP proves browser origin, not browser behavior.

Table of Contents

1. Introduction 3
1.1. The User-Agent Problem 3
1.2. The BVAP Approach 4
1.3. Scope and Non-Goals 4
1.4. Specification Philosophy 5
1.5. Relationship to Existing Code Signing 6
1.6. Relationship to Mutual TLS 6
2. Terminology 7
3. Installation Attestation Model 6
3.1. Overview 6
3.2. Installation Flow 6
3.3. Vendor Attestation Service 7
3.4. BVAP Seal Format (Canonical Payload) 8
3.5. Seal Rotation (Short-Lived Seals) 10
3.6. Revocation Discard 11
3.7. Seal Storage and Embedding 12
4. Transport -- Header Integration 12
4.1. Dedicated Header -- Sec-BVAP (REQUIRED for BVAP use) . . 12
4.2. User-Agent Embedding (Compatibility Only) 13
4.3. Examples 14
5. Vendor Responsibility 15
5.1. The Vendor Principle 15
5.2. Vendor Categories 16
6. Server-Side Handling 17
6.1. Basic Mode (Informational Provenance Signal) 17
6.2. Optional Verification Mode (Cryptographic Assurance) . 18
6.3. Provenance Classification 19
7. Seal Revocation 20
7.1. Revocation by Renewal Denial 20
7.2. Revocation Discard Obligation 21
7.3. Immediate Revocation (Optional) 21
8. Security Considerations 22
8.1. Vendor Trust Assumption (Foundational) 22
8.2. Seal Forgery 23
8.3. Vendor Key Compromise 24
8.4. Seal Cloning and Distributed Reuse 24
8.5. Attestation Service Availability 25
8.6. Build Validation (Vendor-Defined) 26
8.7. Threats Outside BVAP Scope 26
9. Privacy Considerations 27
10. IANA Considerations 28
11. References 28
11.1. Normative References 28
11.2. Informative References 29
Appendix A. Transport Examples 30
Appendix B. Vendor Attestation Service Examples 31
Author's Address 32

1. Introduction

1.1. The User-Agent Problem

Current web architecture contains no standardized mechanism for browser vendors to cryptographically attest that a browser instance genuinely originates from their distributed software.

The HTTP User-Agent field [RFC9110] is informational only and carries no cryptographic authenticity guarantees. As a result:

- Any software may claim any browser identity.
- Browser vendor identity is trivially spoofable.

- Servers cannot distinguish vendor-attested browser software from arbitrary software asserting browser identity.
- Anonymous browser software and falsely claimed browser identity are operationally indistinguishable at the protocol layer.

This represents a software provenance gap in modern web architecture. BVAP addresses this gap by enabling browser vendors to issue cryptographically verifiable attestations for browser software they distribute.

1.2. The BVAP Approach

BVAP addresses this problem at the point of browser installation rather than at the point of each HTTP request.

When a browser is installed, it contacts its vendor’s Attestation Service and presents itself for verification. The vendor confirms that the installation corresponds to a legitimate, unmodified build of their software, and issues a cryptographic Vendor Attestation Seal (BVAP Seal).

The browser stores this Seal and transmits it primarily via the Sec-BVAP HTTP header. The Seal MAY additionally be embedded in the User-Agent string for backward compatibility. Every subsequent HTTP request carries the Seal, allowing servers to identify the vendor that attested this browser installation.

No DNS lookups occur during normal request processing.
No per-request cryptography is required.
Seals are short-lived and periodically renewed (Section 3.5).

This is analogous to a physical inspection seal: the inspector (vendor) examines the product (browser installation), applies a seal, and periodically reissues it. The seal travels with the product but does not last forever.

1.3. Scope and Non-Goals

BVAP is a browser software provenance and vendor attestation protocol. It answers exactly one question:

> "Does this browser instance originate from a vendor-attested browser build?"

BVAP does NOT answer:

- Who the user is.
- Whether the user is human.
- Whether the browser behavior is benign.
- Whether the browser is automated or remotely controlled.
- Whether the endpoint device is compromised.
- Whether the request is fraudulent or abusive.

Behavioral trust, fraud analysis, abuse mitigation, automation detection, and user authentication remain outside the scope of this protocol.

BVAP is NOT:

- A CAPTCHA replacement.
- A human verification system.
- A bot elimination mechanism.
- A fraud prevention system.
- A behavioral trust engine.
- A user identity system.

BVAP intentionally permits:

- Anonymous browsing (no BVAP token required).
- Self-compiled browser builds (Anonymous/Unattested status).
- Open source browser forks (vendor registers independently).
- Research browser environments.
- Enterprise-managed browser deployments.
- Automated operation of legitimate browser software.

Non-Browser HTTP Clients:

BVAP does not attempt to attest generic HTTP clients, API libraries, command-line tools (curl, wget), scripts, mobile embedded agents, IoT devices, or any non-browser HTTP software.

Such clients operate as Anonymous/Unattested when interacting with BVAP-aware servers. They are not penalized for the absence of BVAP attestation. Operators of non-browser HTTP clients MAY voluntarily participate as vendors under their own domain if they wish to provide vendor attestation for their software, but no such participation is required or expected.

For automated agent identity (bots, crawlers, IoT devices), see related work such as the Signed Agent Identity Protocol, which operates on the agent side of the same architectural problem.

BVAP proves browser origin, not browser behavior.

1.4. Specification Philosophy

BVAP is a proposal, not a prescription.

This document specifies the protocol semantics required for interoperability:

- Seal format (Section 3.4).
- Cryptographic signature scheme (Ed25519, Section 3.4.2).
- Transport (Sec-BVAP header, Section 4).
- Verification procedure (Section 3.4.3, Section 6.2).
- DNS-based public key publication (Section 6.2).

These elements MUST be observed by conforming implementations because they define the wire-level contract between browsers, vendors, and servers. Without them, interoperability is impossible.

This document does NOT prescribe:

- How vendors internally validate their browser builds (Section 8.6).
- How vendors operate their Attestation Service infrastructure (subject to operational privacy requirements in Section 3.6).
- How servers interpret BVAP signals beyond the trust classification framework (Section 6.3).
- Which vendors are trustworthy (Section 8.1 -- vendor trust is at the discretion of relying parties).

Vendors and operators retain full operational discretion within the boundaries defined by this specification. The protocol establishes the rules of the game; participants choose how to play within those rules.

This separation between specification (mandatory for interoperability) and implementation (vendor-defined) is

consistent with established IETF practice: protocols define the wire, not the engine room.

1.5. Relationship to Existing Code Signing

A natural question is why BVAP is needed given that browser binaries are already cryptographically signed through platform code signing mechanisms: Authenticode (Windows), codesign (macOS), and various Linux package signing schemes.

These mechanisms validate the browser binary LOCALLY at install or launch time. The operating system verifies the signature and confirms the binary is authentic before allowing execution.

However, this verification is invisible at the HTTP layer:

- The server has no visibility into the local code signing state of the browser making the request.
- Platform code signing signals do not traverse HTTP.
- A bot or scraper pretending to be Chrome cannot fail any code signing check, because there is no code signing check at the server side to begin with.

BVAP exports provenance from the local code signing layer into the HTTP layer. The vendor leverages its existing code signing and release infrastructure (Section 8.6) to issue a Seal that the server can verify, bridging the gap between local software authenticity and remote HTTP visibility.

BVAP is therefore complementary to code signing, not a replacement: local code signing protects the integrity of the binary on the user's device; BVAP communicates that protection to remote servers in a privacy-preserving manner.

1.6. Relationship to Mutual TLS

A second natural question is why BVAP does not use mutual TLS (mTLS) client certificates, an established IETF mechanism for client authentication.

mTLS and BVAP address different problems at different layers of the identity stack:

- Identity axis: mTLS authenticates the USER or organization presenting the certificate. BVAP attests the SOFTWARE making the request. These are orthogonal identity axes -- one identifies WHO, the other identifies WHAT.

- Enrollment model: mTLS requires per-site enrollment. A client certificate is typically issued for a specific service, trust domain, or PKI hierarchy. BVAP requires no enrollment with any server -- a single vendor attestation works with any server that chooses to recognize it.

- User experience: mTLS imposes significant UX overhead. Browser certificate selection dialogs interrupt the user and require active selection. BVAP is invisible to the user and requires no interaction.

- Lifetime and tracking: mTLS certificates are typically long-lived and carry stable identity, enabling cross-site correlation. BVAP Seals rotate every 14-30 days with Revocation Discard (Section 3.6) to prevent long-term tracking by servers or by the vendor itself.

- Revocation: mTLS revocation requires CRL or OCSP

infrastructure. BVAP revocation is achieved through renewal denial (Section 7) with no separate revocation infrastructure required.

- Deployment cost: mTLS deployment requires per-server PKI configuration, trust store management, and client enrollment workflows. BVAP deployment requires only vendor DNS key publication (Section 6.2) and Sec-BVAP header parsing.

BVAP is therefore not a replacement for mTLS. The two mechanisms address different layers of the identity stack and may coexist in the same deployment when both software provenance and user authentication are required.

2. Terminology

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.

Browser Vendor:	Any entity that compiles, signs, and distributes a browser build to end users. Includes commercial organizations, open source projects, and individual developers who distribute browser software.
Vendor Attestation Service:	A service operated by the Browser Vendor that verifies browser installations and issues BVAP Seals. MUST be operated under the vendor’s domain.
BVAP Seal:	A short-lived cryptographic token issued by the Vendor Attestation Service to a verified browser installation. Carried by the browser via Sec-BVAP and periodically renewed.
Attested Browser:	A browser that carries a valid BVAP Seal issued by a recognized vendor.
Unattested Client:	Any HTTP client that does not carry a BVAP Seal. Includes legitimate browsers without BVAP support, privacy-opt-out browsers, automated agents, and HTTP libraries.
Vendor Domain:	The domain under which the vendor operates their Attestation Service and publishes their attestation keys. Examples: google.com, mozilla.org, apple.com.

3. Installation Attestation Model

3.1. Overview

The BVAP attestation model has three phases:

Phase 1 -- Installation:
The browser is installed on the user’s device. At the completion of installation, the browser initiates the attestation process with the Vendor Attestation Service.

Phase 2 -- Attestation:
The vendor verifies that the installation corresponds to a legitimate, unmodified build of their software and issues

a BVAP Seal to the browser.

Phase 3 -- Operation:
The browser stores the BVAP Seal locally and transmits it primarily via the Sec-BVAP header for subsequent HTTP requests. The Seal MAY additionally appear in the User-Agent string for backward compatibility. The Seal is periodically renewed per Section 3.5.

Initial attestation occurs once per installation. Seal renewal occurs periodically (every 14-30 days per Section 3.5) and does not require user action. Attestation MAY also be re-triggered after browser updates at vendor discretion.

3.2. Installation Flow

The following describes the attestation flow at installation:

1. Browser installation completes on the user’s device.

2. The browser generates an Installation Identity:

   install-id = random(256-bit)

   The install-id MUST be:

   - Generated using a cryptographically secure random number generator per [RFC4086].
   - Composed of pure randomness, with NO input derived from device hardware, MAC addresses, hardware IDs, CPU IDs, or any other device-specific value.
   - Unique per installation (per browser profile, per user account on the device).
   - Never reused across reinstallations.

   This design ensures install-id cannot be used for device fingerprinting or cross-installation correlation. An install-id is a fresh, random nonce -- not a derived device identifier.

3. The browser contacts the Vendor Attestation Service:

   POST https://attest.<vendor-domain>/bvap/v1/attest
   Content-Type: application/json

   ```
   {
     "build":      "<browser-name>-<version>",
     "build-hash": "<SHA256 of browser binary>",
     "install-id": "<install-id>",
     "platform":   "<os-platform>",
     "ts":         <unix-timestamp>
   }
   ```

4. The Vendor Attestation Service validates:
   - The build-hash corresponds to a known, signed release.
   - The build version matches the claimed build identifier.
   - The request appears to originate from a fresh installation (rate limiting and anti-abuse checks at vendor discretion).

5. On successful validation, the Service issues a BVAP Seal (Section 3.4) and returns it to the browser.

6. The browser stores the Seal in tamper-evident local storage and transmits it in subsequent HTTP requests primarily via the Sec-BVAP header (Section 4). The Seal is periodically renewed per Section 3.5.

If attestation fails (e.g., tampered binary, network error), the browser operates without a BVAP Seal as an Unattested Client. Attestation failure MUST NOT prevent browser operation.

3.3. Vendor Attestation Service

The Vendor Attestation Service is operated by the Browser Vendor under their domain. It MUST:

- Be reachable at https://attest.<vendor-domain>/bvap/v1/attest during installation and Seal renewal phases.
- Verify that the presented build-hash corresponds to a known, signed release build of the vendor's browser software.
- Issue BVAP Seals only to verified installations.
- Sign Seals using Ed25519 [RFC8032] with the VendorSigningPrivateKey held in hardware-backed secure storage (HSM or equivalent).
- Include an expiry timestamp in every issued Seal (maximum 30 days from issuance).
- Support Seal renewal requests from existing installations.
- Implement Revocation Discard per Section 3.6.
- Publish the VendorSigningPublicKey in DNS for offline server verification:
  _bvap.<vendor-domain>. IN TXT "v=bvap1; pk=<Ed25519-pubkey>"

The Attestation Service MUST NOT:

- Log or retain information that can identify the user.
- Link install-id values across different installations.
- Maintain any mapping between successive Seals issued to the same installation (see Section 3.6).
- Share installation data with third parties.

The Vendor Attestation Service is contacted at installation and at Seal renewal. It is NOT contacted during normal browser operation or per-request processing.

3.4. BVAP Seal Format (Canonical Payload)

The BVAP Seal is a compact, URL-safe string with three colon-separated components. This canonical format eliminates payload reconstruction ambiguity and enables fully offline verification.

Format:

<vendor-domain>:<encoded-claims>:<seal-sig>

Where:

vendor-domain: The vendor's domain (e.g., google.com, mozilla.org). Identifies who issued the Seal and which DNS record to consult for the public key.

encoded-claims: Base64URL-encoded JSON object containing the public attestation claims. This is the payload that the vendor signs.

seal-sig: Base64URL-encoded Ed25519 signature [RFC8032]
           over the canonical signing input (see below).

3.4.1. Encoded Claims Structure

The encoded-claims component is a Base64URL encoding of a compact JSON object containing the following fields:

```
{
  "ver":    "<browser-version>",
  "exp":    <expiry-unix-timestamp>,
  "iat":    <issue-unix-timestamp>
}
```

Field definitions:

ver String. Browser build version identifier as recognized by the vendor (e.g., "chrome-124", "firefox-125"). MUST be present.

exp Integer. Unix timestamp (seconds) at which this Seal expires. MUST NOT exceed iat + 30 days. MUST be present.

iat Integer. Unix timestamp (seconds) at which this Seal was issued. MUST be present.

The JSON object MUST be serialized in canonical form:

- UTF-8 encoded.
- No whitespace between tokens.
- Keys in the order: "ver", "exp", "iat".

The serialized JSON is then Base64URL-encoded (without padding) to produce the encoded-claims component.

Note: The encoded-claims contains ONLY public attestation metadata. It does NOT contain install-id, build-hash, device information, or any user-identifiable data. The install-id used during attestation (Section 3.2) is consumed by the vendor and is NOT transmitted in the Seal.

Future versions of this specification MAY define additional optional claim fields. Implementations MUST ignore unknown claim fields for forward compatibility.

3.4.2. Signing Input

The signature is computed over the concatenation of the vendor domain and the encoded-claims, separated by a single colon:

```
signing-input = <vendor-domain> || ":" || <encoded-claims>
```

This is the exact byte sequence that the vendor signs and that the server reconstructs for verification -- character-for-character identical to what appears in the Sec-BVAP header before the second colon.

```
seal-sig = Base64URL(
             Ed25519.Sign(VendorSigningPrivateKey, signing-input)
           )
```

The Ed25519 signature scheme [RFC8032] is REQUIRED for BVAP. Future versions of this specification MAY define additional signature algorithms identified by a "v" field in encoded-claims.

3.4.3. Verification Procedure

To verify a BVAP Seal, the server:

1. Splits the Seal on ":" into three components:
   vendor-domain, encoded-claims, seal-sig.

2. Retrieves the VendorSigningPublicKey from DNS at
   _bvap.<vendor-domain>. (cached per Section 6.2).

3. Decodes encoded-claims (Base64URL) and parses the JSON.
   Validates that exp > current_time (Seal not expired).

4. Reconstructs the signing input EXACTLY:
   signing-input = <vendor-domain> || ":" || <encoded-claims>

   Note: encoded-claims is used in its original Base64URL form,
   NOT the decoded JSON. This eliminates JSON canonicalization
   ambiguity during verification.

5. Verifies the signature using Ed25519:
   Ed25519.Verify(VendorSigningPublicKey, signing-input, seal-sig)

6. If verification succeeds and Seal is not expired: classify
   as Attested Browser per Section 6.3.

This procedure is fully self-contained. The server does not
need to know install-id, build-hash, or any other vendor-side
state. The encoded-claims provides everything needed.

3.4.4. Example

Conceptual example (signature abbreviated):

```
vendor-domain:    google.com

claims JSON:      {"ver":"chrome-124",
                   "exp":1746748800,
                   "iat":1744156800}

encoded-claims:   eyJ2ZXIiOiJjaHJvbWUtMTI0IiwiZXhwIjoxNzQ2NzQ4ODAw
                  LCJpYXQiOjE3NDQxNTY4MDB9

signing-input:    google.com:eyJ2ZXIiOiJjaHJvbWUtMTI0...

seal-sig:         <Ed25519 signature over signing-input,
                   Base64URL encoded, ~88 characters>

Final Seal:       google.com:eyJ2ZXIiOiJjaHJ...:<seal-sig>

Transmitted in:   Sec-BVAP: google.com:eyJ2ZXIi...:<seal-sig>
```

3.5. Seal Rotation (Short-Lived Seals)

BVAP Seals MUST be short-lived and periodically rotated.

- Seal validity SHOULD NOT exceed 30 days.
- Seal validity of 14 days is RECOMMENDED for privacy-
  sensitive deployments to reduce long-term cross-site
  correlation potential.
- Browsers SHOULD initiate Seal renewal within 5 days before
  expiry by contacting the Vendor Attestation Service.
- Renewal is silent and automatic -- no user action required.
- Seals that have expired MUST be treated as absent by servers.
  Browsers with expired Seals operate as Anonymous/Unattested
  clients until renewal succeeds.

Seal rotation provides three compounding benefits:

Privacy:
Cross-site correlation is limited to the Seal validity window (maximum 30 days) rather than the lifetime of the browser installation. After each rotation, the browser presents a different signature to all sites, breaking any correlation built on the previous Seal.

Revocation:
A compromised or abusive installation is revoked by refusing Seal renewal. No server-side revocation list infrastructure is required. The existing Seal expires naturally within at most 30 days.

Compromise Response:
A leaked or stolen Seal becomes invalid at the next rotation boundary without requiring active propagation of revocation information to servers.

Auto-Renewal Flow:

```
Day 1:    Installation -> Vendor issues Seal #1
          Valid for 30 days. Embedded in User-Agent.

Day 25:   Browser contacts Vendor Attestation Service:
          "My seal expires in 5 days. Issue renewal."
          Vendor checks: is this installation revoked?
            NO  -> Issues Seal #2. Discards Seal #1 record.
            YES -> Refuses. Browser becomes Unattested.

Day 30:   Seal #1 expires. Browser switches to Seal #2.
          No user action. No service interruption.
```

3.6. Revocation Discard (Operational Privacy Obligation)

Revocation Discard is a protocol-level operational privacy requirement imposed on participating vendors. It is NOT a cryptographic guarantee enforced by the protocol itself.

The BVAP protocol cannot cryptographically prove that a vendor has discarded correlation data. Vendors are expected to comply with Revocation Discard as part of their operational obligations when implementing the BVAP Vendor Attestation Service.

This is the same governance model used by Certificate Authorities in PKI: a CA is operationally required to follow its Certificate Practice Statement, even though the protocol cannot enforce CA internal behavior cryptographically.

The Operational Requirement:

Upon issuing a renewed Seal, the Vendor Attestation Service MUST:

- Immediately invalidate the previous Seal.
- Discard records linking the previous Seal to the renewed Seal -- including logs, databases, analytics pipelines, and backup systems -- within a reasonable operational window (RECOMMENDED: 24 hours).
- NOT maintain any mapping between successive Seals issued to the same installation across rotation boundaries.

This commitment ensures that a compliant vendor cannot reconstruct a tracking chain across Seal rotation periods.

A malicious or non-compliant vendor can undermine this property; see Section 8 (Security Considerations) on the vendor trust assumption.

The ONLY persistent record the vendor MAY maintain is a binary installation status: active (eligible for renewal) or revoked (ineligible for renewal). This binary status:

- MUST NOT be linkable to specific Seal values.
- MUST NOT be linkable to user identity or browsing history.
- MUST only be used to determine renewal eligibility.

Revocation Discard + Seal Rotation together provide:

- Server-side privacy: servers cannot track across rotations (different signature).
- Vendor-side privacy: compliant vendors cannot track across rotations (Discard obligation).

The protocol is designed so that BVAP is not a tracking infrastructure when operated by compliant vendors. Non-compliance is a vendor governance issue, not a protocol-level guarantee.

3.7. Seal Storage and Embedding

The BVAP Seal MUST be stored in persistent, tamper-evident storage on the user’s device. Recommended storage locations:

- OS credential store (Windows Credential Manager, macOS Keychain, Linux Secret Service)
- Browser profile storage with integrity verification

The Seal MUST survive browser restarts. Browser updates SHOULD NOT invalidate the Seal unless the vendor explicitly requires re-attestation for that update.

The Seal MUST NOT be shared between different browser profiles or different user accounts on the same device.

4. Transport -- Header Integration

BVAP defines two transport methods for transmitting the Seal: the dedicated Sec-BVAP header (REQUIRED for security use) and User-Agent embedding (informational compatibility only).

Sec-BVAP is REQUIRED for any deployment that relies on BVAP for any operational decision beyond informational logging.

User-Agent embedding is NOT RECOMMENDED for security-sensitive use. It exists solely for backward compatibility with legacy infrastructure that cannot yet process the Sec-BVAP header.

4.1. Dedicated Header -- Sec-BVAP (REQUIRED for BVAP use)

The Sec-BVAP HTTP header is the REQUIRED transport for the BVAP Seal. Any server that intends to act on BVAP signals -- including Optional Verification Mode, traffic classification beyond raw logging, or any policy decision -- MUST receive the Seal via Sec-BVAP.

Header format:

Sec-BVAP: <vendor-domain>:<encoded-claims>:<seal-sig>

The full Seal format including the encoded-claims payload is defined in Section 3.4.

The "Sec-" prefix follows the convention for security-relevant request headers [FETCH]. This prefix:

- Indicates that the header is restricted from being set by JavaScript and other web content.
- Signals to intermediaries that the header is security-relevant and should be handled accordingly.
- Enables CDNs and analytics systems that mask Sec-* headers by default to preserve user privacy.
- Allows cleaner separation from User-Agent semantics.

Browsers implementing BVAP MUST support Sec-BVAP. Browsers MAY additionally include the Seal in the User-Agent string per Section 4.2 for backward compatibility.

Example:

```
GET /api/data HTTP/1.1
Host: example.com
User-Agent: Mozilla/5.0 Chrome/124.0.0.0
Sec-BVAP: google.com:<encoded-claims>:<seal-sig>
```

4.2. User-Agent Embedding (Compatibility Only)

The BVAP Seal MAY also be embedded in the User-Agent string as a product token per [RFC9110] Section 10.1.2. This transport form exists solely for backward compatibility with legacy infrastructure that does not yet support Sec-BVAP.

User-Agent embedding is NOT RECOMMENDED for security-sensitive use. Servers SHOULD prefer Sec-BVAP and SHOULD ignore BVAP tokens in the User-Agent string when Sec-BVAP is also present.

Format:

```
User-Agent: <existing-ua-string> BVAP/<seal>
```

Where <seal> is the full BVAP Seal as defined in Section 3.4. The BVAP token MUST be appended at the end of the User-Agent string, separated by a single space character.

Operational concerns with User-Agent embedding:

- User-Agent values are commonly logged by CDNs, intermediaries, and analytics systems, exposing the Seal to broader visibility than the Sec-BVAP header.
- User-Agent strings are parsed by countless legacy systems that may corrupt, truncate, or misinterpret the BVAP token.
- User-Agent strings are subject to fingerprinting concerns that the Sec-BVAP transport avoids.
- Future protocol evolution is harder when BVAP is bound to User-Agent semantics.

Servers that rely on BVAP for any operational decision MUST use Sec-BVAP. Servers MAY accept User-Agent embedding as a secondary signal during migration periods, but MUST NOT make security decisions based on User-Agent-embedded Seals alone.

4.3. Examples

RECOMMENDED -- Sec-BVAP transport:

```
GET /api/data HTTP/1.1
Host: example.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
  AppleWebKit/537.36 Chrome/124.0.0.0 Safari/537.36
Sec-BVAP: google.com:<encoded-claims>:<seal-sig>
```

NOT RECOMMENDED -- User-Agent embedding (compatibility only):

```
GET /api/data HTTP/1.1
Host: example.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
  AppleWebKit/537.36 Chrome/124.0.0.0 Safari/537.36
  BVAP/google.com:<encoded-claims>:<seal-sig>
```

Firefox with Sec-BVAP:

```
Sec-BVAP: mozilla.org:<encoded-claims>:<seal-sig>
```

Tor Browser with Sec-BVAP:

```
Sec-BVAP: torproject.org:<encoded-claims>:<seal-sig>
```

Browser without BVAP attestation (Anonymous/Unattested):

```
GET /api/data HTTP/1.1
Host: example.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
  AppleWebKit/537.36 Chrome/124.0.0.0 Safari/537.36
[no Sec-BVAP, no BVAP in UA]
```

5. Vendor Responsibility

5.1. The Vendor Principle

Any entity that compiles, signs, and distributes a browser build assumes responsibility for attesting that build.

This principle applies without exception:

- Google is responsible for attesting Chrome builds.
- Mozilla Foundation is responsible for attesting Firefox.
- Apple Inc. is responsible for attesting Safari.
- The Tor Project is responsible for attesting Tor Browser.
- A developer distributing a custom browser fork is responsible for attesting that fork’s builds.

Vendor responsibility is triggered by distribution, not by licensing model. An open source browser is not exempt from vendor responsibility if it is distributed to end users.

A vendor that does not operate a BVAP Attestation Service produces Unattested Clients. This is permitted. Browsers without BVAP attestation are treated as anonymous clients, not as malicious actors.

5.2. Vendor Categories

Commercial vendors:
: Organizations such as Google, Mozilla Foundation, Apple, and Microsoft that distribute browser software at scale. These vendors operate the Attestation Service under their established domain.

Open source projects:
: Projects such as the Tor Project, Brave Software, and Vivaldi that distribute open source or open core browsers.

These projects MUST operate their Attestation Service under their own domain. Being open source does not transfer vendor responsibility to the upstream project.

Example: LibreWolf is a Firefox fork distributed by the LibreWolf Project. LibreWolf operates its own Attestation Service at attest.librewolf.net. LibreWolf browsers carry BVAP/librewolf.net seals, NOT BVAP/mozilla.org seals.

Custom and fork builds:
Developers distributing modified browser builds MUST NOT claim the identity of the upstream vendor in their BVAP token. A custom Chromium fork MUST use its own vendor domain, not google.com.

Enterprise and managed builds:
An enterprise distributing internally modified browser builds to its employees, customers, or controlled environments assumes vendor responsibility for those builds.

Enterprise-managed, kiosk, government-hardened, and internally distributed browser variants MAY participate in BVAP using enterprise-controlled vendor domains.

Example: A bank distributing a hardened Firefox variant to its internal users may operate an Attestation Service at attest.bank.example and issue BVAP/bank.example seals. The bank assumes vendor responsibility for these builds.

This preserves provenance accountability while allowing deployment-specific browser modification, including:
- Pinned versions for compliance environments.
- Removed features for hardened deployments.
- Added enterprise policies and managed configurations.
- Localization or branding variants.

Self-compiled builds:
Users who compile browsers from source for personal use only and do not distribute them are NOT vendors. Their self-compiled browsers operate as Anonymous/Unattested clients. This is intentional and by design.

The freedom to compile, modify, and run browser software from source is a fundamental principle of open source software. BVAP does not restrict this freedom in any way.

A self-compiled browser:
- Operates as Anonymous/Unattested.
- Is NOT penalized or treated as malicious.
- MAY optionally self-register as a vendor under their own domain if they distribute the build to others.

Future work MAY define a Community Attestation model for reproducible open source builds (e.g., via established reproducible build verification infrastructure).

6. Server-Side Handling

6.1. Basic Mode (Informational Provenance Signal)

Basic Mode is INFORMATIONAL ONLY.

Servers MUST treat Basic Mode as unauthenticated. Basic Mode MUST NOT be used for access control decisions.

In Basic Mode, the server reads the BVAP token from the request

and uses it as a provenance signal for traffic classification. No cryptographic verification is performed. The vendor claim in Basic Mode is unauthenticated and trivially spoofable.

Servers operating in Basic Mode MUST NOT grant security-sensitive privileges based solely on the presence of a BVAP token without cryptographic verification. Basic Mode provides provenance signaling and traffic classification benefits, but does NOT provide cryptographic assurance of vendor identity.

Server logic:

1. Parse the request for a BVAP token.
2. If BVAP token present:
   a. Extract vendor-domain from the seal.
   b. Use the vendor claim as an unauthenticated provenance signal.
   c. Apply traffic classification policy.
3. If no BVAP token present:
   a. Treat as Anonymous/Unattested Client.
   b. Apply default anonymous policy.
4. If User-Agent claims a specific browser but no BVAP token is present:
   a. Apply standard anonymous policy.
   b. MAY log as Unverifiable Claim (per Section 6.3).

Basic Mode is appropriate for:

- Traffic classification and routing decisions.
- Distinguishing attested-claiming from non-claiming clients.
- Soft policy (rate limit tiers, log enrichment).
- Telemetry and analytics on browser provenance signaling.

Basic Mode MUST NOT be used for:

- Authentication.
- Authorization or access control.
- Bypassing security controls.
- Granting access to sensitive resources.
- Anti-fraud or abuse prevention enforcement.

Servers requiring any form of access control or cryptographic assurance MUST use Optional Verification Mode (Section 6.2).

6.2. Optional Verification Mode (Cryptographic Assurance)

In Optional Verification Mode, the server cryptographically verifies the BVAP Seal signature using the vendor’s Ed25519 public key retrieved from DNS. This mode provides full cryptographic assurance of vendor identity and Seal authenticity.

This mode is OPT-IN and intended for deployments requiring strong assurance (security-sensitive APIs, regulated services, high-value endpoints).

Verification steps (per Section 3.4.3):

1. Parse the Seal from the Sec-BVAP header:
   <vendor-domain>:<encoded-claims>:<seal-sig>

2. Decode encoded-claims (Base64URL) and parse as JSON.
   Verify exp > current_time. If expired, treat as absent (Anonymous/Unattested).

3. Retrieve the VendorSigningPublicKey from DNS:

_bvap.<vendor-domain>. IN TXT "v=bvap1; pk=<Ed25519-pubkey>"

Cache the result per DNS TTL. DNS lookups MUST occur at most once per vendor-domain per cache TTL period. Servers MUST validate DNSSEC [RFC4033] where available.

4. Reconstruct the signing input exactly:

```
signing-input = <vendor-domain> || ":" || <encoded-claims>
```

Use the encoded-claims component in its original Base64URL form from the Seal -- NOT a re-encoded version. This is the identical byte sequence the vendor signed.

5. Verify the signature:

```
Ed25519.Verify(VendorSigningPublicKey,
               signing-input,
               Base64URL.Decode(seal-sig))
```

6. On successful verification: classify as Attested Browser per Section 6.3.

On verification failure: treat as Anonymous/Unattested. MAY log as potential Seal forgery attempt.

Optional Verification provides cryptographic assurance of:

- Vendor authenticity: only the vendor with the corresponding VendorSigningPrivateKey can produce a valid signature.
- Seal integrity: any modification to vendor-domain, encoded-claims, or the signature itself invalidates verification.
- Non-forgery: without the vendor’s private key, valid signatures cannot be produced for any claims.

Verification is performed entirely locally by the server. The vendor’s Attestation Service is NOT contacted during per-request verification. Only the DNS public key lookup occurs, and this is cached aggressively (RECOMMENDED TTL: 3600 seconds).

DNS Key Pinning (Optional):

Servers MAY pin vendor public keys locally to reduce operational dependency on DNS resolution during verification. Pinned keys:

- MAY be obtained out-of-band from vendor publications, security advisories, or established trust channels.
- SHOULD be periodically refreshed against DNS to detect vendor key rotation events.
- Provide resilience against DNS poisoning, captive portals, resolver compromise, enterprise interception, and other DNS-layer attacks.

When a pinned key disagrees with the current DNS record, servers SHOULD log the discrepancy and apply policy according to their security posture (treat as Anonymous/Unattested, alert administrators, or refresh pin).

Pinning is OPTIONAL. Servers that do not pin keys rely entirely on DNS (RECOMMENDED with DNSSEC validation) for key distribution.

6.3. Provenance Classification

BVAP defines three provenance categories that servers MAY use to inform policy decisions:

Client Type	Condition	Provenance
Attested Browser	Valid BVAP Seal present. Vendor claim accepted (Basic) or verified (Optional).	Verified
Anonymous / Unattested	No BVAP Seal present. OR self-compiled build, OR browser without BVAP, OR user opted out. Honest about its status.	Unknown
Unverifiable Claim	Claims specific browser vendor in UA but no BVAP Seal. Claim without proof.	Claimed but Unverified

The distinction between Anonymous/Unattested and Unverifiable Claim is operationally significant:

- Anonymous/Unattested clients make no false assertions. They are unknown, not dishonest. Default policy applies.

- Unverifiable Claim clients actively assert an identity they cannot prove. This is more concerning than honest anonymity and SHOULD receive reduced trust accordingly.

Provenance classification refers exclusively to confidence in software origin and vendor authenticity. It MUST NOT be interpreted as proof of human presence, benign intent, or absence of automation. Verified Provenance does NOT imply trustworthy behavior -- a Seal-attested browser may still originate from automated, headless, or malware-controlled sessions.

The term "trust" is intentionally avoided in this classification because it carries behavioral semantics. BVAP operates strictly at the provenance layer.

BVAP proves browser origin, not browser behavior.

7. Seal Revocation

BVAP Seal revocation is achieved through Seal rotation denial rather than active revocation list propagation.

7.1. Revocation by Renewal Denial

When the Vendor Attestation Service determines that a browser installation should not be renewed, it refuses to issue a new Seal when the browser requests renewal.

The existing Seal continues to be accepted until its expiry timestamp. After expiry, the browser operates as Anonymous/ Unattested until it successfully obtains a new Seal -- which will not happen for non-renewed installations.

IMPORTANT -- Revocation Scope and Limits:

- Revocation affects ONLY BVAP attestation status. It MUST NOT prevent browser operation, web access, or any other browser functionality.

- A revoked browser continues to function fully as an Anonymous/Unattested client. The user retains full access to the web.
- Revocation does NOT imply maliciousness on the part of the user. A user MUST NOT be characterized as a bad actor on the basis of BVAP revocation status alone.
- Vendors SHOULD publish their revocation criteria transparently. Arbitrary or opaque revocation undermines the trust model.
- Servers MUST NOT use BVAP revocation status as the sole basis for blocking, denying service, or restricting access. BVAP is an informational provenance signal -- not an access control mechanism.

BVAP is not an internet access control system. The protocol provides provenance signaling only. Revocation removes the provenance signal but does not remove the browser, the user, or the right to browse anonymously.

This design eliminates the need for:

- Server-side revocation lists.
- Real-time revocation checking by servers.
- OCSP-style revocation infrastructure.
- Revocation propagation delays.

Revocation is eventual (bounded by Seal validity period) rather than immediate. This is an accepted trade-off for the significant reduction in infrastructure complexity.

7.2. Revocation Discard Obligation

The Revocation Discard commitment defined in Section 3.6 applies throughout the revocation process. Even when revoking an installation, the vendor MUST NOT:

- Link the revoked Seal to any previous Seals.
- Use revocation records to reconstruct browsing history.
- Share revocation data with third parties in a form that identifies the installation or user.

7.3. Immediate Revocation (Optional)

For deployments requiring immediate revocation (before Seal expiry), servers MAY implement optional real-time revocation checking against the vendor’s revocation endpoint:

GET https://attest.<vendor-domain>/bvap/v1/status/<seal-sig>

Response: { "status": "valid" } or { "status": "revoked" }

This endpoint is OPTIONAL for vendors and OPTIONAL for servers. Servers that do not implement real-time revocation checking rely on natural Seal expiry for revocation enforcement.

The seal signature transmitted in this request is opaque and does not reveal user identity. However, vendors MUST NOT log these status check requests in association with site identity, to prevent vendor-side correlation of browsing patterns.

8. Security Considerations

8.1. Vendor Trust Assumption (Foundational)

BVAP fundamentally assumes that browser vendors operate their Attestation Service infrastructure according to the privacy, operational, and security requirements of this specification.

The BVAP trust model relies on ecosystem-level trust in vendors, similar to PKI. No central authority governs vendor behavior.

This includes:

- Honest Seal issuance only to verified browser installations.
- Compliance with Revocation Discard (Section 3.6).
- Secure storage of VendorSigningPrivateKey.
- Honest revocation when warranted by stated criteria.
- No use of attestation infrastructure for user tracking.

A malicious or non-compliant vendor can undermine BVAP privacy and security guarantees independently of the protocol mechanics. Specifically, a non-compliant vendor could:

- Issue Seals to non-browser software (impersonation enablement).
- Maintain hidden mappings between rotated Seals (tracking).
- Share installation data with third parties.
- Refuse Seal renewal arbitrarily or based on opaque criteria.

This trust model is consistent with established PKI governance: Certificate Authorities operate under similar operational trust assumptions, enforced through audit, transparency, and ecosystem accountability rather than cryptographic compulsion. There is no central governing body that compels CA behavior -- trust emerges from ecosystem participation, public accountability, and the practical consequences of misbehavior.

The same applies to BVAP vendors. Trust is established and maintained through:

- Vendor reputation and accountability to their user base.
- Public scrutiny of vendor compliance with this specification.
- Server-side discretion in which vendors to trust.
- User choice of browser vendor.

Servers and users SHOULD evaluate vendor trustworthiness independently. Vendors with poor compliance reputation MAY be treated with reduced trust regardless of valid BVAP attestation. Conversely, a server is never required to trust any particular vendor -- vendor trust is always at the discretion of the relying party.

To make this explicit:

- BVAP does not define a central trust authority.
- BVAP does not establish a vendor approval program.
- BVAP does not maintain a mandatory vendor registry.
- BVAP does not require ecosystem-wide vendor accreditation.

Trust decisions remain entirely local to relying parties. A server is free to trust any subset of vendors, no vendors, or all vendors, based on its own policy. There is no protocol-level entity that compels server trust in any vendor.

This design intentionally avoids reproducing the centralization patterns of established PKI ecosystems. BVAP provides the verification mechanism; relying parties retain full discretion over how that verification is used.

8.2. Seal Forgery

The Seal signature is computed using Ed25519 [RFC8032] with the VendorSigningPrivateKey held exclusively by the vendor. A forger without this private key cannot produce a valid Seal signature for a given vendor-domain.

In Optional Verification Mode (Section 6.2), servers detect forged Seals through Ed25519 signature verification using the vendor's public key retrieved from DNS. This is a cryptographic guarantee: forged signatures cannot pass verification.

In Basic Mode (Section 6.1), the server does not verify the signature and is informationally vulnerable to forged Seals. This is by design: Basic Mode is informational only and MUST NOT be used for security-sensitive decisions.

Servers requiring strong forgery resistance MUST use Optional Verification Mode.

8.3. Vendor Key Compromise

If the VendorSigningPrivateKey is compromised, an attacker can generate valid Seal signatures for any installation under that vendor's domain. This is the primary cryptographic risk in BVAP.

Mitigations:

- VendorSigningPrivateKey MUST be stored in hardware security modules (HSM) with non-exportable key attributes.
- Vendors SHOULD rotate VendorSigningPrivateKey periodically (RECOMMENDED: annually).
- On detected compromise, the vendor MUST:
  - Immediately update the public key in DNS to a new key.
  - Invalidate all Seals signed by the compromised key.
  - Force re-attestation of all active installations.
  - Notify the ecosystem via standard channels (security advisories, browser vendor lists, etc.).
- DNS TTL for _bvap records SHOULD be set to 300-3600 seconds to bound the propagation window for emergency key rotation.
- Servers in Optional Verification Mode automatically reject Seals signed by the old key once the DNS public key has been updated.

8.4. Seal Cloning and Distributed Reuse

BVAP attests to software provenance, not operational execution state or real-time orchestration. A malicious actor may extract a valid BVAP Seal from a legitimate browser installation and distribute it across a high-volume automated botnet infrastructure.

This limitation is a structural property of stateless client-side delivery over the open web. BVAP does not guarantee that the executing agent is free of automation. It only guarantees that the cryptographically bound installation footprint was initially validated by the vendor.

Mitigating the distributed reuse of a cloned Seal falls outside the scope of this protocol. Servers and networks MAY utilize standard rate-limiting, IP reputation profiling, or stateful tracking to detect anomalous, multi-source presentation of

identical signature tokens.

Protocol-level mitigations that DO apply:

- Seal storage in OS credential stores (Section 3.7) reduces extractability from compromised endpoints.
- Seal rotation (Section 3.5) bounds the maximum lifetime of any cloned Seal to the Seal validity window (30 days max, 14 days RECOMMENDED).
- Vendor-side anomaly detection during renewal: if a single install-id is observed renewing from improbable patterns, the vendor MAY refuse renewal (Section 7).
- Optional Verification Mode (Section 6.2) detects forgery but does NOT detect legitimate Seals presented by unauthorized parties.

BVAP intentionally does not attempt to solve the distributed reuse problem at the protocol layer. Solving it would require per-request server interaction with the vendor, undermining the KISS principle and creating significant centralization and privacy concerns. This trade-off is intentional and consistent with the protocol’s scope (Section 1.3).

8.5. Attestation Service Availability

The Vendor Attestation Service is only required during browser installation. If the service is unavailable at install time, the browser operates as an Unattested Client until attestation succeeds. Service unavailability does not affect browsers that have already been attested.

8.6. Build Validation (Vendor-Defined)

This specification defines the attestation transport, verification semantics, and provenance model, but does not mandate how vendors internally validate browser builds.

A vendor MUST validate that a browser instance corresponds to a legitimate, distributed build of their software before issuing a BVAP Seal. The specific mechanism for this validation is vendor-defined and MAY include:

- SHA256 hash matching against a list of known signed release hashes.
- Signed Build Manifest verification with component-level attestation.
- Platform-specific code signing chain validation (Authenticode on Windows, codesign on macOS, etc.).
- Reproducible build verification for open source projects.
- Any combination of the above, or other vendor-specific mechanisms.

The validation mechanism is internal to the Vendor Attestation Service. Different vendors MAY implement different validation approaches based on their build, release, and distribution processes. This flexibility accommodates the operational diversity across commercial vendors, open source projects, enterprise deployments, and custom forks.

Whatever mechanism is chosen, the vendor MUST NOT attest builds they cannot independently verify as their own legitimate distribution.

8.7. Threats Outside BVAP Scope

BVAP does not protect against:

- Automation frameworks controlling a legitimate attested browser (Selenium, Playwright, Puppeteer, etc.).
- Malware controlling an attested browser installation.
- Remote desktop or remote control of a legitimate browser.
- Bot farms using genuine browser installations.

These threats are outside the scope of a software provenance protocol. BVAP proves browser origin, not browser behavior.

9. Privacy Considerations

9.1. User Non-Identification

BVAP MUST NOT enable identification or tracking of individual users. The BVAP Seal carries only:

- The vendor domain (public information).
- An opaque Ed25519 signature that does not contain PII.

The install-id used during attestation is ephemeral and is NOT embedded in the BVAP Seal or the User-Agent string.

9.2. Cross-Site Non-Linkability via Seal Rotation

BVAP Seals rotate every 30 days (Section 3.5). This means:

- The Seal signature changes at each rotation boundary.
- Cross-site correlation using the Seal signature is limited to the current Seal validity window (maximum 30 days).
- After rotation, the browser presents a new signature that cannot be linked to the previous one by any server.

The Revocation Discard obligation (Section 3.6) ensures that even the vendor cannot link successive Seals to reconstruct a long-term browsing profile.

This design provides meaningful privacy protection while preserving vendor attestation functionality. The protocol is not a permanent identifier system.

Servers MUST NOT attempt to link Seal signatures across rotation boundaries for user tracking purposes. Storing Seal signatures in association with user identity or browsing history MUST be considered a violation of this specification.

9.3. Opt-Out

Users MUST be able to disable BVAP attestation in browser settings. When disabled:

- The browser operates as an Unattested Client.
- The BVAP token is NOT included in the User-Agent string.
- Browser functionality MUST NOT be degraded.

Browser vendors MUST NOT penalize users who opt out of BVAP attestation.

9.4. Attestation Service Privacy

The Vendor Attestation Service is contacted once per installation. The service:

- MUST NOT log information that identifies the user.
- MUST NOT retain install-id values after issuing the Seal.
- MUST NOT associate the installation with any account, profile, or identity of the user.

10. IANA Considerations

This document requests IANA to register the following HTTP header field in the "Permanent Message Header Field Names" registry [RFC9110]:

Header field name:	Sec-BVAP
Applicable protocol:	http
Status:	standard
Author/Change controller:	IETF
Specification:	This RFC

This document requests IANA to register the following product token in the HTTP "Product Tokens" registry [RFC9110]:

Product token:	BVAP
Description:	Browser Vendor Attestation Protocol seal (User-Agent backward-compatibility mode). Format: BVAP/<vendor-domain>:<encoded-claims>:<seal-sig>
Specification:	This RFC

This document requests IANA to create the following new registry:

Registry name:	BVAP Vendor Attestation DNS Parameters
Registration policy:	Specification Required
Initial contents:	v, pk as defined in Section 6.2 of this document.

11. References

11.1. Normative References

[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/info/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/info/rfc8174>.

[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>.

[RFC4086] Eastlake 3rd, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, DOI 10.17487/RFC4086, June 2005, <https://www.rfc-editor.org/info/rfc4086>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

11.2. Informative References

[RFC8809] Hodges, J., Jones, J., and M. Jones, "Registries for Web Authentication (WebAuthn)", RFC 8809, DOI 10.17487/RFC8809, August 2020, <https://www.rfc-editor.org/info/rfc8809>.

[RFC9576] Davidson, A., Faz-Hernandez, A., Sullivan, N., and C. Wood, "The Privacy Pass Architecture", RFC 9576, DOI 10.17487/RFC9576, April 2024, <https://www.rfc-editor.org/info/rfc9576>.

[RFC4033] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "DNS Security Introduction and Requirements", RFC 4033, DOI 10.17487/RFC4033, March 2005, <https://www.rfc-editor.org/info/rfc4033>.

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

[FETCH] WHATWG, "Fetch Standard", <https://fetch.spec.whatwg.org/>.

Appendix A. Transport Examples (Informative)

This appendix shows BVAP transport examples for both the REQUIRED Sec-BVAP header (Section 4.1) and the User-Agent backward-compatibility form (Section 4.2).

Note: In these examples:

- <claims> represents Base64URL-encoded JSON claims (e.g., {"ver":"chrome-124","exp":1746748800,"iat":1744156800} -> eyJ2ZXIiOiJjaHJvbWUtMTI0Ii...)
- <sig> represents the 88-character Base64URL Ed25519 signature.

A.1. Attested Browsers (Sec-BVAP, REQUIRED)

Chrome (Windows):

```
GET /api/data HTTP/1.1
Host: example.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
  AppleWebKit/537.36 Chrome/124.0.0.0 Safari/537.36
Sec-BVAP: google.com:<claims>:<sig>
```

Firefox (Linux):

```
GET /api/data HTTP/1.1
Host: example.com
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:125.0)
  Gecko/20100101 Firefox/125.0
Sec-BVAP: mozilla.org:<claims>:<sig>
```

Safari (macOS):

```
Sec-BVAP: apple.com:<claims>:<sig>
```

Tor Browser:

```
Sec-BVAP: torproject.org:<claims>:<sig>
```

LibreWolf (custom fork, own vendor):

```
Sec-BVAP: librewolf.net:<claims>:<sig>
```

Enterprise hardened Firefox (bank example):

```
Sec-BVAP: bank.example:<claims>:<sig>
```

A.2. Attested Browsers (User-Agent backward-compatibility)

Chrome with BVAP in User-Agent (NOT RECOMMENDED for security):

```
GET /api/data HTTP/1.1
Host: example.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
  AppleWebKit/537.36 Chrome/124.0.0.0 Safari/537.36
  BVAP/google.com:<claims>:<sig>
```

Note: Sec-BVAP is REQUIRED for any deployment that relies on BVAP for operational decisions. User-Agent embedding is for backward compatibility only. See Section 4.

A.3. Anonymous / Unattested Clients

Anonymous browser (no claim, no BVAP):

```
GET /api/data HTTP/1.1
Host: example.com
User-Agent: Mozilla/5.0 (X11; Linux x86_64) Custom-Browser/1.0
[No Sec-BVAP, no BVAP in User-Agent. Honest anonymity.]
```

A.4. Unverifiable Claim

Client claiming Chrome identity without BVAP attestation:

```
GET /api/data HTTP/1.1
Host: example.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
  AppleWebKit/537.36 Chrome/124.0.0.0 Safari/537.36
[Claims Chrome but no Sec-BVAP and no BVAP in UA.
 Server treats as Unverifiable Claim per Section 6.3.]
```

Appendix B. Vendor Attestation Service Examples (Informative)

B.1. Attestation Request

```
POST https://attest.google.com/bvap/v1/attest
Content-Type: application/json

{
  "build":      "chrome-124",
  "build-hash": "sha256:a3f8c2d1e9b7...",
  "install-id": "<256-bit random nonce, Base64URL encoded>",
  "platform":   "windows-x64",
  "ts":         1744200000
}
```

B.2. Attestation Response (Success)

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

{
  "seal":       "google.com:<base64url-encoded-claims>:
                 <base64url-signature>",
  "claims":     {
    "ver":      "chrome-124",
    "exp":      1746748800,
    "iat":      1744156800
```

```
  },
  "signature-alg": "ed25519",
  "status":     "attested"
}
```

The "seal" field contains the complete canonical BVAP Seal as defined in Section 3.4. The browser stores this Seal and transmits it in subsequent requests via Sec-BVAP header (Section 4.1). The "claims" field is shown decoded for clarity and is identical to the Base64URL-decoded encoded-claims embedded in the seal.

B.3. Attestation Response (Failure)

```
HTTP/1.1 403 Forbidden
Content-Type: application/json

{
  "status":  "rejected",
  "reason":  "unknown-build-hash",
  "detail":  "Presented build hash does not correspond
              to any known signed release."
}
```

Browser operates as Anonymous/Unattested Client.
MUST NOT retry more than 3 times within 24 hours.

Author’s Address

Srecko Jovancevic
SKGO, IKT Support
Makedonska 22
11000 Belgrade
Serbia

Email: srecko.jovancevic@skgo.org
Email: srecko.jovancevic@gmail.com
URI: https://github.com/sreckojovancevic