Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 25, 2020

J. Yasskin
K. Ueno
Google

July 24, 2019

Signed HTTP Exchanges Implementation Checkpoints
draft-yasskin-httpbis-origin-signed-exchanges-impl-03

Abstract

This document describes checkpoints of draft-yasskin-http-origin-signed-responses to synchronize implementation between clients, intermediates, and publishers.

Note to Readers

Discussion of this draft takes place on the HTTP working group mailing list (ietf-http-wg@w3.org), which is archived at https://lists.w3.org/Archives/Public/ietf-http-wg/ [1].

The source code and issues list for this draft can be found in https://github.com/WICG/webpackage [2].

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 January 25, 2020.

Table of Contents

1. Introduction

Each version of this document describes a checkpoint of [I-D.yasskin-http-origin-signed-responses] that can be implemented in sync by clients, intermediates, and publishers. It defines a technique to detect which version each party has implemented so that mismatches can be detected up-front.

2. Terminology

Absolute URL A string for which the URL parser [3] ([URL]), when run without a base URL, returns a URL rather than a failure, and for which that URL has a null fragment. This is similar to the absolute-URL string [4] concept defined by ([URL]) but might not include exactly the same strings.

Author The entity that wrote the content in a particular resource. This specification deals with publishers rather than authors.

Publisher The entity that controls the server for a particular origin [RFC6454]. The publisher can get a CA to issue certificates for their private keys and can run a TLS server for their origin.

Exchange (noun) An HTTP request URL, content negotiation information, and an HTTP response. This are encoded into the dedicated format in Section 5.3, which uses [I-D.ietf-httpbis-variants-05] to encode the content negotiation information. This is not quite the same meaning as defined by Section 8 of [RFC7540], which assumes the content negotiation information is embedded into HTTP request headers.

Intermediate An entity that fetches signed HTTP exchanges from a publisher or another intermediate and forwards them to another intermediate or a client.

Client An entity that uses a signed HTTP exchange and needs to be able to prove that the publisher vouched for it as coming from its claimed origin.

Unix time Defined by [POSIX] section 4.16 [5].

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.

3. Signing an exchange

In the response of an HTTP exchange the server MAY include a "Signature" header field (Section 3.1) holding a list of one or more parameterised signatures that vouch for the content of the exchange. Exactly which content the signature vouches for can depend on how the exchange is transferred (Section 5).

The client categorizes each signature as "valid" or "invalid" by validating that signature with its certificate or public key and other metadata against the exchange's URL, response headers, and content (Section 3.5). This validity then informs higher-level protocols.

Each signature is parameterised with information to let a client fetch assurance that a signed exchange is still valid, in the face of revoked certificates and newly-discovered vulnerabilities. This assurance can be bundled back into the signed exchange and forwarded to another client, which won't have to re-fetch this validity information for some period of time.

3.1. The Signature Header

The "Signature" header field conveys a single signature for an exchange, accompanied by information about how to determine the authority of and refresh that signature. Each signature directly signs the exchange's URL and response headers and identifies one of those headers that enforces the integrity of the exchange's payload.

The "Signature" header is a Structured Header as defined by [I-D.ietf-httpbis-header-structure-10]. Its value MUST be a parameterised list (Section 3.4 of [I-D.ietf-httpbis-header-structure-10]), and the list MUST contain exactly one element. Its ABNF is:

Signature = sh-param-list

The parameterised identifier in the list MUST have parameters named "sig", "integrity", "validity-url", "date", "expires", "cert-url", and "cert-sha256". This specification gives no meaning to the identifier itself, which can be used as a human-readable identifier for the signature. The present parameters MUST have the following values:

"sig" Byte sequence (Section 3.10 of [I-D.ietf-httpbis-header-structure-10]) holding the signature of most of these parameters and the exchange's URL and response headers.

"integrity" A string (Section 3.8 of [I-D.ietf-httpbis-header-structure-10]) containing a "/"-separated sequence of names starting with the lowercase name of the response header field that guards the response payload's integrity. The meaning of subsequent names depends on the response header field, but for the "digest" header field, the single following name is the name of the digest algorithm that guards the payload's integrity.

"cert-url" A string (Section 3.8 of [I-D.ietf-httpbis-header-structure-10]) containing an absolute URL (Section 2) with a scheme of "https" or "data".

"cert-sha256" Byte sequence (Section 3.10 of [I-D.ietf-httpbis-header-structure-10]) holding the SHA-256 hash of the first certificate found at "cert-url".

"validity-url" A string (Section 3.8 of [I-D.ietf-httpbis-header-structure-10]) containing an absolute URL (Section 2) with a scheme of "https".

"date" and "expires" An integer (Section 3.6 of [I-D.ietf-httpbis-header-structure-10]) representing a Unix time.

The "cert-url" parameter is _not_ signed, so intermediates can update it with a pointer to a cached version.

3.1.1. Examples

The following header is included in the response for an exchange with effective request URI "https://example.com/resource.html". Newlines are added for readability.

```
Signature:
 sig1;

sig=*MEUCIQDXlI2gN3RNBlgFiuRNFpZXcDIaUpX6HIEwcZEc0cZYLAIga9DsVOMM+g5YpwEBdGW3sS+bvnmAJJiSMwhuBdc
  integrity="digest/mi-sha256-03";
  validity-url="https://example.com/resource.validity.1511128380";
  cert-url="https://example.com/oldcerts";
  cert-sha256=*W7uB969dFW3Mb5ZefPS9Tq5ZbH5iSmOILpjv2qEArmI=*;
  date=1511128380; expires=1511733180
```

The signature uses a secp256r1 certificate within "https://example.com/".

It relies on the "Digest" response header with the mi-sha256-03 digest algorithm to guard the integrity of the response payload.

The signature includes a "validity-url" that includes the first time the resource was seen. This allows multiple versions of a resource at the same URL to be updated with new signatures, which allows clients to avoid transferring extra data while the old versions don't have known security bugs.

The certificate at "https://example.com/certs" has a "subjectAltName" of "example.com", meaning that if it and its signature validate, the exchange can be trusted as having an origin of "https://example.com/".

3.2. CBOR representation of exchange response headers

To sign an exchange's response headers, they need to be serialized into a byte string. Since intermediaries and distributors might rearrange, add, or just reserialize headers, we can't use the literal bytes of the headers as this serialization. Instead, this section defines a CBOR representation that can be embedded into other CBOR, canonically serialized (Section 3.4), and then signed.

The CBOR representation of a set of response metadata and headers is the CBOR ([RFC7049]) map with the following mappings:

- The byte string ':status' to the byte string containing the response's 3-digit status code, and

- For each response header field, the header field's lowercase name as a byte string to the header field's value as a byte string.

3.2.1. Example

Given the HTTP exchange:

```
GET / HTTP/1.1
Host: example.com
Accept: */*

HTTP/1.1 200
Content-Type: text/html
Digest: mi-sha256-03=dcRDgR2GM35DluAV13PzgnG6+pvQwPywfFvAu1UeFrs=
Signed-Headers: "content-type", "digest"

<!doctype html>
<html>
...
```

The cbor representation consists of the following item, represented using the extended diagnostic notation from [CDDL] appendix G:

```
{
  'digest': 'mi-sha256-03=dcRDgR2GM35DluAV13PzgnG6+pvQwPywfFvAu1UeFrs=',
  ':status': '200',
  'content-type': 'text/html'
}
```

3.3. Loading a certificate chain

The resource at a signature's "cert-url" MUST have the "application/cert-chain+cbor" content type, MUST be canonically-encoded CBOR (Section 3.4), and MUST match the following CDDL:

```
cert-chain = [
  "&#128220;&#9939;", ; U+1F4DC U+26D3
  + {
    cert: bytes,
    ? ocsp: bytes,
    ? sct: bytes,
    * tstr => any,
  }
]
```

The first map (second item) in the CBOR array is treated as the end-entity certificate, and the client will attempt to build a path ([RFC5280]) to it from a trusted root using the other certificates in the chain.

1. Each "cert" value MUST be a DER-encoded X.509v3 certificate ([RFC5280]). Other key/value pairs in the same array item define properties of this certificate.

2. The first certificate's "ocsp" value MUST be a complete, DER-encoded OCSP response for that certificate (using the ASN.1 type "OCSPResponse" defined in [RFC6960]). Subsequent certificates MUST NOT have an "ocsp" value.

3. Each certificate's "sct" value if any MUST be a "SignedCertificateTimestampList" for that certificate as defined by Section 3.3 of [RFC6962].

Loading a "cert-url" takes a "forceFetch" flag. The client MUST:

1. Let "raw-chain" be the result of fetching ([FETCH]) "cert-url". If "forceFetch" is _not_ set, the fetch can be fulfilled from a cache using normal HTTP semantics [RFC7234]. If this fetch fails, return "invalid".

2. Let "certificate-chain" be the array of certificates and properties produced by parsing "raw-chain" using the CDDL above. If any of the requirements above aren't satisfied, return "invalid". Note that this validation requirement might be impractical to completely achieve due to certificate validation implementations that don't enforce DER encoding or other standard constraints.

3. Return "certificate-chain".

3.4. Canonical CBOR serialization

Within this specification, the canonical serialization of a CBOR item uses the following rules derived from Section 3.9 of [RFC7049] with erratum 4964 applied:

- Integers and the lengths of arrays, maps, and strings MUST use the smallest possible encoding.

- Items MUST NOT be encoded with indefinite length.

- The keys in every map MUST be sorted in the bytewise lexicographic order of their canonical encodings. For example, the following keys are correctly sorted:

  1. 10, encoded as 0A.

  2. 100, encoded as 18 64.

  3. -1, encoded as 20.

  4. "z", encoded as 61 7A.

  5. "aa", encoded as 62 61 61.

  6. [100], encoded as 81 18 64.

  7. [-1], encoded as 81 20.

  8. false, encoded as F4.

Note: this specification does not use floating point, tags, or other more complex data types, so it doesn't need rules to canonicalize those.

3.5. Signature validity

The client MUST parse the "Signature" header field as the parameterised list (Section 4.2.5 of [I-D.ietf-httpbis-header-structure-10]) described in Section 3.1. If an error is thrown during this parsing or any of the requirements described there aren't satisfied, the exchange has no valid signatures. Otherwise, each member of this list represents a signature with parameters.

The client MUST use the following algorithm to determine whether each signature with parameters is invalid or potentially-valid for an exchange's

- "requestUrl", a byte sequence that can be parsed into the exchange's effective request URI (Section 5.5 of [RFC7230]),

- "responseHeaders", a byte sequence holding the canonical serialization (Section 3.4) of the CBOR representation (Section 3.2) of the exchange's response metadata and headers, and

- "payload", a stream of bytes constituting the exchange's payload body (Section 3.3 of [RFC7230]). Note that the payload body is the message body with any transfer encodings removed.

Potentially-valid results include:

- The signed headers of the exchange so that higher-level protocols can avoid relying on unsigned headers, and

- Either a certificate chain or a public key so that a higher-level protocol can determine whether it's actually valid.

This algorithm accepts a "forceFetch" flag that avoids the cache when fetching URLs. A client that determines that a potentially-valid certificate chain is actually invalid due to an expired OCSP response MAY retry with "forceFetch" set to retrieve an updated OCSP from the original server.

1. Let:

    * "signature" be the signature (byte sequence in the parameterised identifier's "sig" parameter).

    * "integrity" be the signature's "integrity" parameter.

    * "validity-url" be the signature's "validity-url" parameter.

* "cert-url" be the signature's "cert-url" parameter, if any.

* "cert-sha256" be the signature's "cert-sha256" parameter, if any.

* "date" be the signature's "date" parameter, interpreted as a Unix time.

* "expires" be the signature's "expires" parameter, interpreted as a Unix time.

2. Set "publicKey" and "signing-alg" depending on which key fields are present:

    1. Assert: "cert-url" is present.

        1. Let "certificate-chain" be the result of loading the certificate chain at "cert-url" passing the "forceFetch" flag (Section 3.3). If this returns "invalid", return "invalid".

        2. Let "main-certificate" be the first certificate in "certificate-chain".

        3. Set "publicKey" to "main-certificate"'s public key.

        4. If "publicKey" is an RSA key, return "invalid".

        5. If "publicKey" is a key using the secp256r1 elliptic curve, set "signing-alg" to ecdsa_secp256r1_sha256 as defined in Section 4.2.3 of [TLS1.3].

        6. Otherwise, return "invalid".

3. If "expires" is more than 7 days (604800 seconds) after "date", return "invalid".

4. If the current time is before "date" or after "expires", return "invalid".

5. Let "message" be the concatenation of the following byte strings. This matches the [TLS1.3] format to avoid cross-protocol attacks if anyone uses the same key in a TLS certificate and an exchange-signing certificate.

    1. A string that consists of octet 32 (0x20) repeated 64 times.

2. A context string: the ASCII encoding of "HTTP Exchange 1 b3".

   Note: As this is a snapshot of a draft of [I-D.yasskin-http-origin-signed-responses], it uses a distinct context string.

3. A single 0 byte which serves as a separator.

4. If "cert-sha256" is set, a byte holding the value 32 followed by the 32 bytes of the value of "cert-sha256". Otherwise a 0 byte.

5. The 8-byte big-endian encoding of the length in bytes of "validity-url", followed by the bytes of "validity-url".

6. The 8-byte big-endian encoding of "date".

7. The 8-byte big-endian encoding of "expires".

8. The 8-byte big-endian encoding of the length in bytes of "requestUrl", followed by the bytes of "requestUrl".

9. The 8-byte big-endian encoding of the length in bytes of "responseHeaders", followed by the bytes of "responseHeaders".

6. If "cert-url" is present and the SHA-256 hash of "main-certificate"'s "cert_data" is not equal to "cert-sha256" (whose presence was checked when the "Signature" header field was parsed), return "invalid".

   Note that this intentionally differs from TLS 1.3, which signs the entire certificate chain in its Certificate Verify (Section 4.4.3 of [TLS1.3]), in order to allow updating the stapled OCSP response without updating signatures at the same time.

7. If "signature" is not a valid signature of "message" by "publicKey" using "signing-alg", return "invalid".

8. If "headers", interpreted according to Section 3.2, does not contain a "Content-Type" response header field (Section 3.1.1.5 of [RFC7231]), return "invalid".

   Clients MUST interpret the signed payload as this specified media type instead of trying to sniff a media type from the bytes of the payload, for example by attaching an "X-Content-

Type-Options: nosniff" header field ([FETCH]) to the extracted response.

9. If "integrity" does not match "digest/mi-sha256-03", return "invalid".

10. If "payload" doesn't match the integrity information in the header described by "integrity", return "invalid".

11. Return "potentially-valid" with "certificate-chain".

Note that the above algorithm can determine that an exchange's headers are potentially-valid before the exchange's payload is received. Similarly, if "integrity" identifies a header field and parameter like "Digest: mi-sha256-03" ([I-D.thomson-http-mice]) that can incrementally validate the payload, early parts of the payload can be determined to be potentially-valid before later parts of the payload. Higher-level protocols MAY process parts of the exchange that have been determined to be potentially-valid as soon as that determination is made but MUST NOT process parts of the exchange that are not yet potentially-valid. Similarly, as the higher-level protocol determines that parts of the exchange are actually valid, the client MAY process those parts of the exchange and MUST wait to process other parts of the exchange until they too are determined to be valid.

3.6. Updating signature validity

Both OCSP responses and signatures are designed to expire a short time after they're signed, so that revoked certificates and signed exchanges with known vulnerabilities are distrusted promptly.

This specification provides no way to update OCSP responses by themselves. Instead, clients need to re-fetch the "cert-url" (Section 3.5, Paragraph 6) to get a chain including a newer OCSP response.

The "validity-url" parameter (Paragraph 5) of the signatures provides a way to fetch new signatures or learn where to fetch a complete updated exchange.

Each version of a signed exchange SHOULD have its own validity URLs, since each version needs different signatures and becomes obsolete at different times.

The resource at a "validity-url" is "validity data", a CBOR map matching the following CDDL ([CDDL]):

```
validity = {
  ? signatures: [ + bytes ]
  ? update: {
    ? size: uint,
  }
]
```

The elements of the "signatures" array are parameterised identifiers (Section 4.2.6 of [I-D.ietf-httpbis-header-structure-10]) meant to replace the signatures within the "Signature" header field pointing to this validity data. If the signed exchange contains a bug severe enough that clients need to stop using the content, the "signatures" array MUST NOT be present.

If the the "update" map is present, that indicates that a new version of the signed exchange is available at its effective request URI (Section 5.5 of [RFC7230]) and can give an estimate of the size of the updated exchange ("update.size"). If the signed exchange is currently the most recent version, the "update" SHOULD NOT be present.

If both the "signatures" and "update" fields are present, clients can use the estimated size to decide whether to update the whole resource or just its signatures.

3.6.1. Examples

For example, say a signed exchange whose URL is "https://example.com/resource" has the following "Signature" header field (with line breaks included and irrelevant fields omitted for ease of reading).

```
Signature:
 sig1;
  sig=*MEUCIQ...*;
  ...
  validity-url="https://example.com/resource.validity.1511157180";
  cert-url="https://example.com/oldcerts";
  date=1511128380; expires=1511733180
```

At 2017-11-27 11:02 UTC, "sig1" has expired, so the client needs to fetch "https://example.com/resource.validity.1511157180" (the "validity-url" of "sig1") if it wishes to update that signature. This URL might contain:

```
{
  "signatures": [
    'sig1; '
    'sig=*MEQCIC/I9Q+7BZFP6cSDsWx43pBAL0ujTbON/
+7RwKVk+ba5AiB3FSFLZqpzmDJ0NumNwN04pqgJZE99fcK86UjkPbj4jw==*; '
    'validity-url="https://example.com/resource.validity.1511157180"; '
    'integrity="digest/mi-sha256-03"; '
    'cert-url="https://example.com/newcerts"; '
    'cert-sha256=*J/lEm9kNRODdCmINbvitpvdYKNQ+YgBj99DlYp4fEXw=*; '
    'date=1511733180; expires=1512337980'
  ],
  "update": {
    "size": 5557452
  }
}
```

This indicates that the client could fetch a newer version at "https://example.com/resource" (the original URL of the exchange), or that the validity period of the old version can be extended by replacing the original signature with the new signature provided. The signature of the updated signed exchange would be:

```
Signature:
 sig1;
  sig=*MEQCIC...*;
  ...
  validity-url="https://example.com/resource.validity.1511157180";
  cert-url="https://example.com/newcerts";
  date=1511733180; expires=1512337980
```

3.7. The Accept-Signature header

The "Accept-Signature" request header is not used.

4. Cross-origin trust

To determine whether to trust a cross-origin exchange, the client takes a "Signature" header field (Section 3.1) and the exchange's

- "requestUrl", a byte sequence that can be parsed into the exchange's effective request URI (Section 5.5 of [RFC7230]),
- "responseHeaders", a byte sequence holding the canonical serialization (Section 3.4) of the CBOR representation (Section 3.2) of the exchange's response metadata and headers, and
- "payload", a stream of bytes constituting the exchange's payload body (Section 3.3 of [RFC7230]).

The client MUST parse the "Signature" header into a list of signatures according to the instructions in Section 3.5, and run the following algorithm for each signature, stopping at the first one that returns "valid". If any signature returns "valid", return "valid". Otherwise, return "invalid".

1. If the signature's "validity-url" parameter (Paragraph 5) is not same-origin [6] with "requestUrl", return "invalid".

2. Use Section 3.5 to determine the signature's validity for "requestUrl", "responseHeaders", and "payload", getting "certificate-chain" back. If this returned "invalid" or didn't return a certificate chain, return "invalid".

3. Let "response" be the response metadata and headers parsed out of "responseHeaders".

4. If Section 3 of [RFC7234] forbids a shared cache from storing "response", return "invalid".

5. If "response"'s headers contain an uncached header field, as defined in Section 4.1, return "invalid".

6. Let "authority" be the host component of "requestUrl".

7. Validate the "certificate-chain" using the following substeps. If any of them fail, re-run Section 3.5 once over the signature with the "forceFetch" flag set, and restart from step 2. If a substep fails again, return "invalid".

    1. Use "certificate-chain" to validate that its first entry, "main-certificate" is trusted as "authority"'s server certificate ([RFC5280] and other undocumented conventions). Let "path" be the path that was used from the "main-certificate" to a trusted root, including the "main-certificate" but excluding the root.

    2. Validate that "main-certificate" has the CanSignHttpExchanges extension (Section 4.2).

    3. Validate that either "main-certificate" has a Validity Period no longer than 90 days, or that the current date is 2019-08-01 or before and "main-certificate" was issued on 2019-05-01 or before.

    4. Validate that "main-certificate" has an "ocsp" property (Section 3.3) with a valid OCSP response whose lifetime ("nextUpdate - thisUpdate") is less than 7 days ([RFC6960]).

Note that this does not check for revocation of intermediate certificates, and clients SHOULD implement another mechanism for that.

5. Validate that valid SCTs from trusted logs are available from any of:

   + The "SignedCertificateTimestampList" in "main-certificate"'s "sct" property (Section 3.3),

   + An OCSP extension in the OCSP response in "main-certificate"'s "ocsp" property, or

   + An X.509 extension in the certificate in "main-certificate"'s "cert" property,

   as described by Section 3.3 of [RFC6962].

8. Return "valid".

4.1. Uncached header fields

Hop-by-hop and other uncached headers MUST NOT appear in a signed exchange. These will eventually be listed in [I-D.ietf-httpbis-cache], but for now they're listed here:

- Hop-by-hop header fields listed in the Connection header field (Section 6.1 of [RFC7230]).
- Header fields listed in the no-cache response directive in the Cache-Control header field (Section 5.2.2.2 of [RFC7234]).
- Header fields defined as hop-by-hop:
  - Connection
  - Keep-Alive
  - Proxy-Connection
  - Trailer
  - Transfer-Encoding
  - Upgrade
- Stateful headers as defined below.

4.1.1. Stateful header fields

As described in Section 6.1 of [I-D.yasskin-http-origin-signed-responses], a publisher can cause problems if they sign an exchange that includes private information. There's no way for a client to be sure an exchange does or does not include private information, but header fields that store or convey stored state in the client are a good sign.

A stateful response header field modifies state, including authentication status, in the client. The HTTP cache is not considered part of this state. These include but are not limited to:

- "Authentication-Control", [RFC8053]
- "Authentication-Info", [RFC7615]
- "Clear-Site-Data", [W3C.WD-clear-site-data-20171130]
- "Optional-WWW-Authenticate", [RFC8053]
- "Proxy-Authenticate", [RFC7235]
- "Proxy-Authentication-Info", [RFC7615]
- "Public-Key-Pins", [RFC7469]
- "Sec-WebSocket-Accept", [RFC6455]
- "Set-Cookie", [RFC6265]
- "Set-Cookie2", [RFC2965]
- "SetProfile", [W3C.NOTE-OPS-OverHTTP]
- "Strict-Transport-Security", [RFC6797]
- "WWW-Authenticate", [RFC7235]

4.2. Certificate Requirements

We define a new X.509 extension, CanSignHttpExchanges to be used in the certificate when the certificate permits the usage of signed exchanges. When this extension is not present the client MUST NOT accept a signature from the certificate as proof that a signed exchange is authoritative for a domain covered by the certificate. When it is present, the client MUST follow the validation procedure in Section 4.

```
   CanSignHttpExchanges ::= NULL
```

Note that this extension contains an ASN.1 NULL (bytes "05 00") because some implementations have bugs with empty extensions.

Leaf certificates without this extension need to be revoked if the private key is exposed to an unauthorized entity, but they generally don't need to be revoked if a signing oracle is exposed and then removed.

CA certificates, by contrast, need to be revoked if an unauthorized entity is able to make even one unauthorized signature.

Certificates with this extension MUST be revoked if an unauthorized entity is able to make even one unauthorized signature.

Starting 2019-05-01, certificates with this extension MUST have a Validity Period no greater than 90 days.

Conforming CAs MUST NOT mark this extension as critical.

Starting 2019-05-01, a conforming CA MUST NOT issue certificates with this extension unless, for each dNSName in the subjectAltName extension of the certificate to be issued:

1. An "issue" or "issuewild" CAA property ([RFC6844]) exists that authorizes the CA to issue the certificate; and

2. The "cansignhttpexchanges" parameter (Section 4.2.1) is present on the property and is equal to "yes"

Clients MUST NOT accept certificates with this extension in TLS connections (Section 4.4.2.2 of [TLS1.3]).

This draft of the specification identifies the CanSignHttpExchanges extension with the id-ce-canSignHttpExchangesDraft OID:

```
id-ce-google OBJECT IDENTIFIER ::= { 1 3 6 1 4 1 11129 }
id-ce-canSignHttpExchangesDraft OBJECT IDENTIFIER ::= { id-ce-google 2 1
22 }
```

This OID might or might not be used as the final OID for the extension, so certificates including it might need to be reissued once the final RFC is published.

Some certificates have already been issued with this extension and with validity periods longer than 90 days. These certificates will not immediately be treated as invalid. Instead:

- Clients MUST reject certificates with this extension that were issued after 2019-05-01 and have a Validity Period longer than 90 days.

- After 2019-08-01, clients MUST reject all certificates with this extension that have a Validity Period longer than 90 days.

4.2.1. Extensions to the CAA Record: cansignhttpexchanges Parameter

A CAA parameter "cansignhttpexchanges" is defined for the "issue" and "issuewild" properties defined by [RFC6844]. The value of this parameter, if specified, MUST be "yes".

5. Transferring a signed exchange

A signed exchange can be transferred in several ways, of which three are described here.

5.1. Same-origin response

Same-origin responses are not implemented.

5.2. HTTP/2 extension for cross-origin Server Push

Cross origin push is not implemented.

5.3. application/signed-exchange format

To allow signed exchanges to be the targets of "<link rel=prefetch>" tags, we define the "application/signed-exchange" content type that represents a signed HTTP exchange, including a request URL, response metadata and header fields, and a response payload.

When served over HTTP, a response containing an "application/signed-exchange" payload MUST include at least the following response header fields, to reduce content sniffing vulnerabilities:

- Content-Type: application/signed-exchange;v=_version_

- X-Content-Type-Options: nosniff

This content type consists of the concatenation of the following items:

1. 8 bytes consisting of the ASCII characters "sxg1-b3" followed by a 0 byte, to serve as a file signature. This is redundant with the MIME type, and recipients that receive both MUST check that

they match and, if they don't, either stop parsing or redirect to the "fallbackUrl" in the next two entries.

Note: As this is a snapshot of a draft of [I-D.yasskin-http-origin-signed-responses], it uses a distinct file signature.

2. 2 bytes storing a big-endian integer "fallbackUrlLength".

3. "fallbackUrlLength" bytes holding a "fallbackUrl", which MUST UTF-8 decode to an absolute URL with a scheme of "https".

   Note: The byte location of the fallback URL is intended to remain invariant across versions of the "application/signed-exchange" format so that parsers encountering unknown versions can always find a URL to redirect to.

4. 3 bytes storing a big-endian integer "sigLength". If this is larger than 16384 (16*1024), parsing MUST fail.

5. 3 bytes storing a big-endian integer "headerLength". If this is larger than 524288 (512*1024), parsing MUST fail.

6. "sigLength" bytes holding the "Signature" header field's value (Section 3.1).

7. "headerLength" bytes holding "signedHeaders", the canonical serialization (Section 3.4) of the CBOR representation of the response headers of the exchange represented by the "application/signed-exchange" resource (Section 3.2), excluding the "Signature" header field.

8. The payload body (Section 3.3 of [RFC7230]) of the exchange represented by the "application/signed-exchange" resource.

   Note that the use of the payload body here means that a "Transfer-Encoding" header field inside the "application/signed-exchange" header block has no effect. A "Transfer-Encoding" header field on the outer HTTP response that transfers this resource still has its normal effect.

5.3.1. Cross-origin trust in application/signed-exchange

To determine whether to trust a cross-origin exchange stored in an "application/signed-exchange" resource, pass the "Signature" header field's value, "fallbackUrl" as the effective request URI, "signedHeaders", and the payload body to the algorithm in Section 4.

5.3.2. Content negotiation

If the signed response headers include a "Variants-04" header field, the client MUST use the cache behavior algorithm in Section 4 of [I-D.ietf-httpbis-variants-05] to check that the signed response is an appropriate representation for the request the client is trying to fulfil. If the response is not an appropriate representation, the client MUST treat the signature as invalid. Note the mismatch between the name of the header field and the version of the Variants draft.

5.3.3. Example

An example "application/signed-exchange" file representing a possible signed exchange with https://example.com/ [7] follows, with lengths represented by descriptions in "<>"s, CBOR represented in the extended diagnostic format defined in Appendix G of [CDDL], and most of the "Signature" header field and payload elided with a ...:

```
sxg1-b3\0<2-byte length of the following url string>
https://example.com/<3-byte length of the following header
value><3-byte length of the encoding of the
following map>sig1; sig=*...; integrity="digest/mi-sha256-03"; ...{
    ':status': '200',
    'content-type': 'text/html'
}<!doctype html>\r\n<html>...
```

6. Security considerations

All of the security considerations from Section 6 of [I-D.yasskin-http-origin-signed-responses] apply.

7. Privacy considerations

Normally, when a client fetches "https://o1.com/resource.js", "o1.com" learns that the client is interested in the resource. If "o1.com" signs "resource.js", "o2.com" serves it as "https://o2.com/o1resource.js", and the client fetches it from there, then "o2.com" learns that the client is interested, and if the client executes the Javascript, that could also report the client's interest back to "o1.com".

Often, "o2.com" already knew about the client's interest, because it's the entity that directed the client to "o1resource.js", but there may be cases where this leaks extra information.

For non-executable resource types, a signed response can improve the privacy situation by hiding the client's interest from the original publisher.

To prevent network operators other than "o1.com" or "o2.com" from learning which exchanges were read, clients SHOULD only load exchanges fetched over a transport that's protected from eavesdroppers. This can be difficult to determine when the exchange is being loaded from local disk, but when the client itself requested the exchange over a network it SHOULD require TLS ([TLS1.3]) or a successor transport layer, and MUST NOT accept exchanges transferred over plain HTTP without TLS.

8. IANA considerations

This depends on the following IANA registrations in [I-D.yasskin-http-origin-signed-responses]:

- The "Signature" header field
- The application/cert-chain+cbor media type

This document also modifies the registration for:

8.1. Internet Media Type application/signed-exchange

Type name: application

Subtype name: signed-exchange

Required parameters:

- v: A string denoting the version of the file format. ([RFC5234] ABNF: "version = DIGIT/%x61-7A") The version defined in this specification is "b3". When used with the "Accept" header field (Section 5.3.1 of [RFC7231]), this parameter can be a comma (,)-separated list of version strings. ([RFC5234] ABNF: "version-list = version *("," version)") The server is then expected to reply with a resource using a particular version from that list.

  Note: As this is a snapshot of a draft of [I-D.yasskin-http-origin-signed-responses], it uses a distinct version number.

Magic number(s): 73 78 67 31 2D 62 33 00

The other fields are the same as the registration in [I-D.yasskin-http-origin-signed-responses].

9. References

9.1. Normative References

[CDDL] Birkholz, H., Vigano, C., and C. Bormann, "Concise Data Definition Language (CDDL): A Notational Convention to Express Concise Binary Object Representation (CBOR) and JSON Data Structures", RFC 8610, DOI 10.17487/RFC8610, June 2019, <https://www.rfc-editor.org/info/rfc8610>.

[FETCH] WHATWG, "Fetch", July 2019, <https://fetch.spec.whatwg.org/>.

[I-D.ietf-httpbis-header-structure-10] Nottingham, M. and P. Kamp, "Structured Headers for HTTP", draft-ietf-httpbis-header-structure-10 (work in progress), April 2019, <https://tools.ietf.org/html/draft-ietf-httpbis-header-structure-10>.

[I-D.ietf-httpbis-variants-05] Nottingham, M., "HTTP Representation Variants", draft-ietf-httpbis-variants-05 (work in progress), March 2019, <https://tools.ietf.org/html/ietf-httpbis-variants-05>.

[I-D.yasskin-http-origin-signed-responses] Yasskin, J., "Signed HTTP Exchanges", draft-yasskin-http-origin-signed-responses-06 (work in progress), July 2019.

[POSIX] IEEE and The Open Group, "The Open Group Base Specifications Issue 7", name IEEE, value 1003.1-2008, 2016 Edition, 2016, <http://pubs.opengroup.org/onlinepubs/9699919799/basedefs/>.

[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>.

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <https://www.rfc-editor.org/info/rfc5234>.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <https://www.rfc-editor.org/info/rfc5280>.

[RFC6844] Hallam-Baker, P. and R. Stradling, "DNS Certification Authority Authorization (CAA) Resource Record", RFC 6844, DOI 10.17487/RFC6844, January 2013, <https://www.rfc-editor.org/info/rfc6844>.

[RFC6960] Santesson, S., Myers, M., Ankney, R., Malpani, A., Galperin, S., and C. Adams, "X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP", RFC 6960, DOI 10.17487/RFC6960, June 2013, <https://www.rfc-editor.org/info/rfc6960>.

[RFC6962] Laurie, B., Langley, A., and E. Kasper, "Certificate Transparency", RFC 6962, DOI 10.17487/RFC6962, June 2013, <https://www.rfc-editor.org/info/rfc6962>.

[RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <https://www.rfc-editor.org/info/rfc7049>.

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <https://www.rfc-editor.org/info/rfc7230>.

[RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <https://www.rfc-editor.org/info/rfc7231>.

[RFC7234] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Caching", RFC 7234, DOI 10.17487/RFC7234, June 2014, <https://www.rfc-editor.org/info/rfc7234>.

[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>.

[TLS1.3] 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>.

[URL] WHATWG, "URL", July 2019, <https://url.spec.whatwg.org/>.

9.2. Informative References

[I-D.ietf-httpbis-cache]
Fielding, R., Nottingham, M., and J. Reschke, "HTTP Caching", draft-ietf-httpbis-cache-05 (work in progress), July 2019.

[I-D.thomson-http-mice]
Thomson, M. and J. Yasskin, "Merkle Integrity Content Encoding", draft-thomson-http-mice-03 (work in progress), August 2018.

[I-D.yasskin-http-origin-signed-responses-03]
Yasskin, J., "Signed HTTP Exchanges", draft-yasskin-http-origin-signed-responses-03 (work in progress), March 2018, <https://tools.ietf.org/html/draft-yasskin-http-origin-signed-responses-03>.

[I-D.yasskin-http-origin-signed-responses-04]
Yasskin, J., "Signed HTTP Exchanges", draft-yasskin-http-origin-signed-responses-04 (work in progress), June 2018, <https://tools.ietf.org/html/draft-yasskin-http-origin-signed-responses-04>.

[I-D.yasskin-http-origin-signed-responses-05]
Yasskin, J., "Signed HTTP Exchanges", draft-yasskin-http-origin-signed-responses-05 (work in progress), January 2019, <https://tools.ietf.org/html/draft-yasskin-http-origin-signed-responses-05>.

[RFC2965] Kristol, D. and L. Montulli, "HTTP State Management Mechanism", RFC 2965, DOI 10.17487/RFC2965, October 2000, <https://www.rfc-editor.org/info/rfc2965>.

[RFC6265] Barth, A., "HTTP State Management Mechanism", RFC 6265, DOI 10.17487/RFC6265, April 2011, <https://www.rfc-editor.org/info/rfc6265>.

[RFC6454] Barth, A., "The Web Origin Concept", RFC 6454, DOI 10.17487/RFC6454, December 2011, <https://www.rfc-editor.org/info/rfc6454>.

[RFC6455] Fette, I. and A. Melnikov, "The WebSocket Protocol", RFC 6455, DOI 10.17487/RFC6455, December 2011, <https://www.rfc-editor.org/info/rfc6455>.

[RFC6797] Hodges, J., Jackson, C., and A. Barth, "HTTP Strict Transport Security (HSTS)", RFC 6797, DOI 10.17487/RFC6797, November 2012, <https://www.rfc-editor.org/info/rfc6797>.

[RFC7235] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Authentication", RFC 7235, DOI 10.17487/RFC7235, June 2014, <https://www.rfc-editor.org/info/rfc7235>.

[RFC7469] Evans, C., Palmer, C., and R. Sleevi, "Public Key Pinning Extension for HTTP", RFC 7469, DOI 10.17487/RFC7469, April 2015, <https://www.rfc-editor.org/info/rfc7469>.

[RFC7540] Belshe, M., Peon, R., and M. Thomson, Ed., "Hypertext Transfer Protocol Version 2 (HTTP/2)", RFC 7540, DOI 10.17487/RFC7540, May 2015, <https://www.rfc-editor.org/info/rfc7540>.

[RFC7615] Reschke, J., "HTTP Authentication-Info and Proxy-Authentication-Info Response Header Fields", RFC 7615, DOI 10.17487/RFC7615, September 2015, <https://www.rfc-editor.org/info/rfc7615>.

[RFC8053] Oiwa, Y., Watanabe, H., Takagi, H., Maeda, K., Hayashi, T., and Y. Ioku, "HTTP Authentication Extensions for Interactive Clients", RFC 8053, DOI 10.17487/RFC8053, January 2017, <https://www.rfc-editor.org/info/rfc8053>.

[W3C.NOTE-OPS-OverHTTP] Hensley, P., Metral, M., Shardanand, U., Converse, D., and M. Myers, "Implementation of OPS Over HTTP", W3C NOTE NOTE-OPS-OverHTTP, June 1997.

[W3C.WD-clear-site-data-20171130] West, M., "Clear Site Data", World Wide Web Consortium WD WD-clear-site-data-20171130, November 2017, <https://www.w3.org/TR/2017/WD-clear-site-data-20171130>.

9.3. URIs

[1] https://lists.w3.org/Archives/Public/ietf-http-wg/

[2] https://github.com/WICG/webpackage

[3] https://url.spec.whatwg.org/#concept-url-parser

[4] https://url.spec.whatwg.org/#absolute-url-string

[5] http://pubs.opengroup.org/onlinepubs/9699919799/basedefs/V1_chap04.html#tag_04_16

[6] https://html.spec.whatwg.org/multipage/origin.html#same-origin

[7] https://example.com/

Appendix A. Change Log

draft-03

Vs. draft-02

- Updates to match [I-D.yasskin-http-origin-signed-responses-05].
- UTF-8 decode the fallback URL.
- Define a CAA parameter to opt into certificate issuance, which CAs need to enforce after May 1.
- Limit lifetimes of certificates issued after May 1 to 90 days.
- Accept-Signature and same-origin responses are removed.

Vs. [I-D.yasskin-http-origin-signed-responses-05]:

- Versions in file signatures and context strings are "b3".
- Signed exchanges can only be transmitted in the application/signed-exchange format, not HTTP/2 Push or plain HTTP request/response pairs.
- The Accept-Signature request header isn't used.
- Removed non-normative sections.
- Only 1 signature is supported.
- Removed support for ed25519 signatures.
- The above UTF-8 decoding.
- The above CAA parameter and certificate lifetimes.
- Versioned the Variants header field at draft-ietf-httpbis-variants-05 (but spelled Variants-04) and the mi-sha256 digest algorithm at draft-thomson-http-mice-03.

- Allow mismatches between the MIME type and file signature to redirect to the fallback URL.

draft-02

Vs. draft-01:

- Define absolute URLs, and limit the schemes each instance can use.
- Update to mice-03 including the Digest header.
- Define the "integrity" field of the Signature header to include the digest algorithm.
- Put a fallback URL at the beginning of the "application/signed-exchange" format, and remove ':url' key from the CBOR representation of the exchange's request and response metadata and headers.
- The new signed message format which embeds the exact bytes of the CBOR representation of the exchange's request and response metadata and headers.
- When validating the signature validity, move the "payload" integrity check steps to after verifying "header".
- Versions in file signatures and context strings are "b2".

draft-01

Vs. [I-D.yasskin-http-origin-signed-responses-04]:

- The MI header and mi-sha256 content-encoding are renamed to MI-Draft2 and mi-sha256-draft2 in case [I-D.thomson-http-mice] changes.
- Signed exchanges cannot be transmitted using HTTP/2 Push.
- Removed non-normative sections.
- The mi-sha256 encoding must have records <= 16kB.
- The signature must be <=16kB long.
- The HTTP request and response headers together must be <=512kB.
- Versions in file signatures and context strings are "b1".

- Only 1 signature is supported.
- Removed support for ed25519 signatures.

draft-00

Vs. [I-D.yasskin-http-origin-signed-responses-03]:

- Removed non-normative sections.
- Only 1 signature is supported.
- Only 2048-bit RSA keys are supported.
- The certificate chain resource uses the TLS 1.3 Certificate message format rather than a CBOR-based format.
- OCSP responses and SCTs are not checked.
- Certificates without the CanSignHttpExchanges extension are allowed.
- The signature string starts with 64 0x20 octets like the TLS 1.3 signature format.
- The application/http-exchange+cbor format is replaced with a more specialized application/signed-exchange format.
- Signed exchanges can only be transmitted using the application/signed-exchange format, not HTTP/2 Push or plain HTTP request/response pairs.
- Only the MI payload-integrity header is supported.
- The mi-sha256 encoding must have records <= 16kB.
- The Accept-Signature header isn't used.
- Require absolute URLs.

Appendix B. Acknowledgements

Thanks to Andrew Ayer, Devin Mullins, Ilari Liusvaara, Justin Schuh, Mark Nottingham, Mike Bishop, Ryan Sleevi, and Yoav Weiss for comments that improved this draft.

Authors' Addresses

Jeffrey Yasskin
Google

Email: jyasskin@chromium.org

Kouhei Ueno
Google

Email: kouhei@chromium.org