Network Working Group J. Yasskin
Internet-Draft K. Ueno
Intended status: Standards Track Google
Expires: October 8, 2018 April 06, 2018

# Signed HTTP Exchanges Implementation Checkpoints
# draft-yasskin-httpbis-origin-signed-exchanges-impl-00

## Abstract

This document describes checkpoints of [I-D.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

## Copyright Notice

Table of Contents

1. Introduction . . . . . . . . . . . . . . . . . . . . . . . . 3
2. Terminology . . . . . . . . . . . . . . . . . . . . . . . . . 3
3. Signing an exchange . . . . . . . . . . . . . . . . . . . . . 3
  3.1. The Signature Header . . . . . . . . . . . . . . . . . . 4
    3.1.1. Examples . . . . . . . . . . . . . . . . . . . . . . 5
    3.1.2. Open Questions . . . . . . . . . . . . . . . . . . . 5
  3.2. CBOR representation of exchange headers . . . . . . . . . 6
    3.2.1. Example . . . . . . . . . . . . . . . . . . . . . . . 6
  3.3. Loading a certificate chain . . . . . . . . . . . . . . . 7
  3.4. Canonical CBOR serialization . . . . . . . . . . . . . . 8
  3.5. Signature validity . . . . . . . . . . . . . . . . . . . 9
    3.5.1. Open Questions . . . . . . . . . . . . . . . . . . . 12
  3.6. Updating signature validity . . . . . . . . . . . . . . . 12
    3.6.1. Examples . . . . . . . . . . . . . . . . . . . . . . 13
  3.7. The Accept-Signature header . . . . . . . . . . . . . . . 14
4. Cross-origin trust . . . . . . . . . . . . . . . . . . . . . 14
  4.1. Stateful header fields . . . . . . . . . . . . . . . . . 15
  4.2. Certificate Requirements . . . . . . . . . . . . . . . . 16
5. Transferring a signed exchange . . . . . . . . . . . . . . . 16
  5.1. Same-origin response . . . . . . . . . . . . . . . . . . 16
  5.2. HTTP/2 extension for cross-origin Server Push . . . . . . 16
  5.3. application/signed-exchange format . . . . . . . . . . . 16
6. Security considerations . . . . . . . . . . . . . . . . . . . 17
7. Privacy considerations . . . . . . . . . . . . . . . . . . . 17
8. IANA considerations . . . . . . . . . . . . . . . . . . . . . 18
  8.1. Internet Media Type application/signed-exchange . . . . . 18
9. References . . . . . . . . . . . . . . . . . . . . . . . . . 19
  9.1. Normative References . . . . . . . . . . . . . . . . . . 19
  9.2. Informative References . . . . . . . . . . . . . . . . . 21
  9.3. URIs . . . . . . . . . . . . . . . . . . . . . . . . . . 22
Appendix A. Change Log . . . . . . . . . . . . . . . . . . . . . 22
Appendix B. Acknowledgements . . . . . . . . . . . . . . . . . . 23
Authors' Addresses . . . . . . . . . . . . . . . . . . . . . . . 23

## 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

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/response pair. This can either be a request from a client and the matching response from a server or the request in a PUSH_PROMISE and its matching response stream. Defined by Section 8 of [RFC7540].

Intermediate An entity that fetches signed HTTP exchanges from an 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 [3].

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 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 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-02]. Its value MUST be a list (Section 4.8 of [I-D.ietf-httpbis-header-structure-02]) of parameterised labels (Section 4.4 of [I-D.ietf-httpbis-header-structure-02]), and the list MUST contain exactly one element.

Each parameterised label MUST have parameters named "sig", "integrity", "validityUrl", "date", and "expires". Each parameterised label MUST also have "certUrl" and "certSha256" parameters. This specification gives no meaning to the label itself, which can be used as a human-readable identifier for the signature (see Section 3.1.2, Paragraph 1). The present parameters MUST have the following values:

"sig"  Binary content (Section 4.5 of [I-D.ietf-httpbis-header-structure-02]) holding the signature of most of these parameters and the exchange's headers.

"integrity"  A string (Section 4.2 of [I-D.ietf-httpbis-header-structure-02]) containing the lowercase name of the response header field that guards the response payload's integrity.

"certUrl"  A string (Section 4.2 of [I-D.ietf-httpbis-header-structure-02]) containing an absolute-URL string [4] ([URL]).

"certSha256"  Binary content (Section 4.5 of [I-D.ietf-httpbis-header-structure-02]) holding the SHA-256 hash of the first certificate found at "certUrl".

"validityUrl"  A string (Section 4.2 of [I-D.ietf-httpbis-header-structure-02]) containing an absolute-URL string [5] ([URL]).

"date" and "expires"  An unsigned integer (Section 4.1 of [I-D.ietf-httpbis-header-structure-02]) representing a Unix time.

The "certUrl" 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=*t7LoYw6vwL2FSZRNJPYdNdYjfZSQkaCQeqpBD1whcy/
6AAamVJ2OryXoXv6ACVBQgPV13o5de9oOVcOGGMX9fsf2ve1UDw/
ITpeimB7n3zcuDEePzIcPbUnicicN2yodZAfr5il7BBJTs8L+V2ZERI16nJfrOZOvUfhvuUaMDGQXx5StIj7XLiX7/
caxPz5ctwglgVAwCmoVPhmYFLq391O+hEssHSk2xkY6r/D9V2cKMikBBOTZ+JFyrnS/
f2B4li7YASIY0YX64ifCmCw97cQTngXax6Upoie44IAe+6JngOie9JlDgcMF3YZ1uxNGWl9VwlalSwWgi1YA9Ff7mQ;
  integrity="mi";
  validityUrl="https://example.com/resource.validity.1511128380";
  certUrl="https://example.com/certs";
  certSha256=*W7uB969dFW3Mb5ZefPS9Tq5ZbH5iSmOILpjv2qEArmI;
  date=1511128380; expires=1511733180
```

The signatures uses a 2048-bit RSA certificate within "https://example.com/".

It relies on the "MI" response header to guard the integrity of the response payload.

The signature includes a "validityUrl" 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/".

### 3.1.2. Open Questions

[I-D.ietf-httpbis-header-structure-02] provides a way to parameterise labels but not other supported types like binary content.  If the "Signature" header field is notionally a list of parameterised signatures, maybe we should add a "parameterised binary content" type.

Should the certUrl and validityUrl be lists so that intermediates can offer a cache without losing the original URLs?  Putting lists in dictionary fields is more complex than [I-D.ietf-httpbis-header-structure-02] allows, so they're single items for now.

## 3.2. CBOR representation of exchange headers

To sign an exchange's 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 an exchange "exchange"'s headers is the CBOR ([RFC7049]) array with the following content:

1.  The map mapping:

    * The byte string ':method' to the byte string containing "exchange"'s request's method.

    * The byte string ':url' to the byte string containing "exchange"'s request's effective request URI, which MUST be an absolute-URL string [6] ([URL]).

    * For each request header field in "exchange", the header field's lowercase name as a byte string to the header field's value as a byte string.

2.  The map mapping:

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

    * for each response header field in "exchange", 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 https://example.com/ HTTP/1.1
Accept: */*

HTTP/1.1 200
Content-Type: text/html
Content-Encoding: mi-sha256
MI: mi-sha256=20addcf7368837f616d549f035bf6784ea6d4bf4817a3736cd2fc7a763897fe3

<0x0000000000004000><!doctype html>
<html>
...
```

The cbor representation consists of the following item, represented using the extended diagnostic notation from [I-D.ietf-cbor-cddl] appendix G:

```
[
  {
    ':url': 'https://example.com/'
    ':method': 'GET',
  },
  {
    'mi': 'mi-
sha256=20addcf7368837f616d549f035bf6784ea6d4bf4817a3736cd2fc7a763897fe3',
    ':status': '200',
    'content-type': 'text/html'
    'content-encoding': 'mi-sha256',
  }
]
```

## 3.3. Loading a certificate chain

The resource at a signature's "certUrl" MUST contain a TLS 1.3 Certificate message (Section 4.4.2 of [I-D.ietf-tls-tls13]) containing X.509v3 certificates.

Parsing notes:

1. This resource MUST NOT include the 4-byte header that would appear in a Handshake message.

2. Since this fetch is not in response to a CertificateRequest, the certificate_request_context MUST be empty, and a non-empty value MUST cause the parse to fail.

The client MUST ignore unknown or unexpected extensions.

Loading a "certUrl" takes a "forceFetch" flag.  The client MUST:

1. Let "raw-chain" be the result of fetching ([FETCH]) "certUrl". 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" as the TLS Certificate message as described 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 list of parameterised values (Section 4.8.1 of [I-D.ietf-httpbis-header-structure-02]) 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". 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.

1. Let "payload" be the payload body (Section 3.3 of [RFC7230]) of "exchange". Note that the payload body is the message body with any transfer encodings removed.

2. Let:

   * "signature" be the signature (binary content in the parameterised label's "sig" parameter).

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

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

   * "certUrl" be the signature's "certUrl" parameter, if any.

   * "certSha256" be the signature's "certSha256" 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.

3. If "integrity" names a header field other than "MI" ([I-D.thomson-http-mice]) or this header field is not present in "exchange"'s response headers or which the client cannot use to check the integrity of "payload" (for example, the header field is new and hasn't been implemented yet), then return "invalid". Clients MUST be able to check the integrity of "payload" using the "MI" ([I-D.thomson-http-mice]) header field.

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

    1. Assert: "certUrl" is present.

        1. Let "certificate-chain" be the result of loading the certificate chain at "certUrl" 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 not a 2048-bit RSA public key, return "invalid".

        5. The client MUST define a partial function from public key types to signing algorithms, and this function must at the minimum include the following mappings:

            RSA, 2048 bits: rsa_pss_rsae_sha256 or rsa_pss_pss_sha256, as defined in Section 4.2.3 of [I-D.ietf-tls-tls13], depending on which of the rsaEncryption OID or RSASSA-PSS OID [RFC8017] is used.

            Set "signing-alg" to the result of applying this function to the type of "main-certificate"'s public key. If the function is undefined on this input, return "invalid".

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

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

7. Let "message" be the concatenation of the following byte strings. This matches the [I-D.ietf-tls-tls13] format to avoid cross-protocol attacks when TLS certificates are used to sign manifests.

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

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

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

    4. The bytes of the canonical CBOR serialization (Section 3.4) of a CBOR map mapping:

        1. If "certSha256" is set:

            1. The text string "certSha256" to the byte string value of "certSha256".

        2. The text string "validityUrl" to the byte string value of "validityUrl".

        3. The text string "date" to the integer value of "date".

        4. The text string "expires" to the integer value of "expires".

        5. The text string "headers" to the CBOR representation (Section 3.2) of "exchange"'s headers.

8. If "certUrl" is present and the SHA-256 hash of "main-certificate"'s "cert_data" is not equal to "certSha256" (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 [I-D.ietf-tls-tls13]), in order to allow updating the stapled OCSP response without updating signatures at the same time. Note that this difference doesn't matter for this version of this draft since OCSP responses aren't checked.

9. If "signature" is a valid signature of "message" by "publicKey" using "signing-alg", return "potentially-valid" with "certificate-chain". Otherwise, return "invalid".

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 like "MI" ([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.

#### 3.5.1.  Open Questions

Should the signed message use the TLS format (with an initial 64 spaces) even though these certificates can't be used in TLS servers?

### 3.6.  Updating signature validity

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.

The "validityUrl" 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 "validityUrl" is "validity data", a CBOR map matching the following CDDL ([I-D.ietf-cbor-cddl]):

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

The elements of the "signatures" array are parameterised labels (Section 4.4 of [I-D.ietf-httpbis-header-structure-02]) 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...;
  ...
  validityUrl="https://example.com/resource.validity.1511157180";
  certUrl="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 "validityUrl" of "sig1") to update that signatures. This URL might contain:

```
{
  "signatures": [
    'sig1; '
    'sig=*MEQCIC/I9Q+7BZFP6cSDsWx43pBAL0ujTbON/
+7RwKVk+ba5AiB3FSFLZqpzmDJ0NumNwN04pqgJZE99fcK86UjkPbj4jw; '
    'validityUrl="https://example.com/resource.validity.1511157180"; '
    'integrity="mi"; '
    'certUrl="https://example.com/newcerts"; '
    'certSha256=*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...;
  ...
  validityUrl="https://example.com/resource.validity.1511157180";
  certUrl="https://example.com/newcerts";
  date=1511733180; expires=1512337980
```

## 3.7. The Accept-Signature header

This section isn't implemented.

## 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". 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 "validityUrl" parameter (Paragraph 5) is not same-origin [7] with "exchange"'s effective request URI (Section 5.5 of [RFC7230]), return "invalid".

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

3. If "exchange"'s request method is not safe (Section 4.2.1 of [RFC7231]) or not cacheable (Section 4.2.3 of [RFC7231]), return "invalid".

4. If "exchange"'s headers contain a stateful header field, as defined in Section 4.1, return "invalid".

5. Let "authority" be the host component of "exchange"'s effective request URI.

6. 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.

7. Return "valid".

## 4.1. 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 request header field informs the server of per-client state. These include but are not limited to:

- "Authorization", [RFC7235]
- "Cookie", [RFC6265]
- "Cookie2", [RFC2965]
- "Proxy-Authorization", [RFC7235]
- "Sec-WebSocket-Key", [RFC6455]

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]
- "Optional-WWW-Authenticate", [RFC8053]
- "Proxy-Authenticate", [RFC7235]
- "Proxy-Authentication-Info", [RFC7615]
- "Sec-WebSocket-Accept", [RFC6455]
- "Set-Cookie", [RFC6265]

- "Set-Cookie2", [RFC2965]

- "SetProfile", [W3C.NOTE-OPS-OverHTTP]

- "WWW-Authenticate", [RFC7235]

### 4.2. Certificate Requirements

For this draft, no new X.509 extension is required.

## 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

Receiving a Signature header as part of a normal HTTP exchange is 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 parse a resource with content type "application/signed-exchange;v=b0", the client MUST run the following algorithm:

Read 3 bytes and interpret them as a big-endian integer "headerLength".

If "headerLength" is larger than 524288 (512kB), parsing MUST fail.

Read "headerLength" bytes, and parse them as a CBOR item.  If this item isn't canonically encoded (Section 3.4) or doesn't match the following CDDL, parsing MUST fail:

```
signed-exchange-header = [
  { ':method': bytes,
    ':url': bytes,
    * bytes => bytes,
  },
  { ':status': bytes,
    'signature': bytes,
    * bytes => bytes,
  },
]
```

The first element of the array is interpreted as the exchange's request headers with lowercase names, with the request method in the ':method' key's value, and the effective request URI, which MUST be an absolute-URL string [8] ([URL]), in the ':url' key's value.

The second element of the array is interpreted as the exchange's response headers with lowercase names, with the 3-digit response status code in the ':status' key's value.

If any header field name includes uppercase characters, parsing MUST fail.

Pass the "Signature" response header and the exchange with that header removed to the algorithm in Section 4. Fail if this returns "invalid".

The remainder of the resource is the exchange's payload, encoded with the "mi-sha256" content encoding ([I-D.thomson-http-mice]). If the "mi-sha256" record length (the first 8 bytes of the payload) is greater than 16kB, or if any of the integrity proofs fail validation, parsing MUST fail.

## 6. Security considerations

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

In addition, because this draft does not check for certificate revocation and allows signatures from certificates that can be used in normal TLS servers with no defense against future-dated signatures, clients MUST NOT trust signed exchanges as authoritative for their claimed origin without some explicit opt-in by their user.

## 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 ([I-D.ietf-tls-tls13]) 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 registration in [I-D.yasskin-http-origin-signed-responses]:

o  The "Signature" header field

This document also registers:

### 8.1. Internet Media Type application/signed-exchange

Type name: application

Subtype name: signed-exchange

Required parameters:

o  v: A string denoting the version of the file format. ([RFC5234] ABNF: "version = DIGIT/%x61-7A") The version defined in this specification is "b0". 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 does not use a simple integer to describe its version.

Optional parameters: N/A

Encoding considerations: binary

Security considerations: see Section 6.6 of [I-D.yasskin-http-origin-signed-responses]

Interoperability considerations: N/A

Published specification: This specification (see Section 5.3).

Applications that use this media type: N/A

Fragment identifier considerations: N/A

Additional information:

Deprecated alias names for this type: N/A

Magic number(s): 82 A?

File extension(s): .sxg

Macintosh file type code(s): N/A

Person and email address to contact for further information: See Authors' Addresses section.

Intended usage: COMMON

Restrictions on usage: N/A

Author: See Authors' Addresses section.

Change controller: IESG

## 9. References

### 9.1. Normative References

[FETCH]    WHATWG, "Fetch", April 2018, <https://fetch.spec.whatwg.org/>.

[HTML]    WHATWG, "HTML", April 2018, <https://html.spec.whatwg.org/multipage>.

[I-D.ietf-cbor-cddl]
Birkholz, H., Vigano, C., and C. Bormann, "Concise data definition language (CDDL): a notational convention to express CBOR data structures", draft-ietf-cbor-cddl-02 (work in progress), February 2018.

[I-D.ietf-httpbis-header-structure-02]
Nottingham, M. and P. Kamp, "Structured Headers for HTTP", draft-ietf-httpbis-header-structure-02 (work in progress), November 2017, <https://tools.ietf.org/html/draft-ietf-httpbis-header-structure-02>.

[I-D.ietf-tls-tls13]
Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", draft-ietf-tls-tls13-28 (work in progress), March 2018.

[I-D.thomson-http-mice]
Thomson, M., "Merkle Integrity Content Encoding", draft-thomson-http-mice-02 (work in progress), October 2016.

[I-D.yasskin-http-origin-signed-responses]
Yasskin, J., "Signed HTTP Exchanges", draft-yasskin-http-origin-signed-responses-03 (work in progress), March 2018.

[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>.

[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>.

[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>.

[RFC8017]  Moriarty, K., Ed., Kaliski, B., Jonsson, J., and A. Rusch, "PKCS #1: RSA Cryptography Specifications Version 2.2", RFC 8017, DOI 10.17487/RFC8017, November 2016, <https://www.rfc-editor.org/info/rfc8017>.

[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>.

[URL]  WHATWG, "URL", April 2018, <https://url.spec.whatwg.org/>.

## 9.2. Informative References

[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>.

[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>.

[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.

## 9.3. URIs

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

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

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

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

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

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

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

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

## Appendix A. Change Log

draft-00

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

o  Removed non-normative sections.

o  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

Authors' Addresses

Jeffrey Yasskin
Google

Email: jyasskin@chromium.org

Kouhei Ueno
Google

Email: kouhei@chromium.org