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 21, 2019.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents
Table of Contents

1. Introduction ............................................. 3
2. Terminology .............................................. 3
3. Signing an exchange ...................................... 4
   3.1. The Signature Header ............................... 4
   3.1.1. Examples ........................................ 5
   3.2. CBOR representation of exchange headers .......... 6
   3.2.1. Example .......................................... 7
   3.3. Loading a certificate chain ........................ 7
   3.4. Canonical CBOR serialization ..................... 9
   3.5. Signature validity ................................ 9
   3.6. Updating signature validity ....................... 12
   3.6.1. Examples ........................................ 13
   3.7. The Accept-Signature header ..................... 14
   3.7.1. Integrity identifiers .......................... 15
   3.7.2. Key type identifiers .......................... 16
   3.7.3. Key value identifiers .......................... 16
   3.7.4. Examples ........................................ 16
4. Cross-origin trust ....................................... 17
   4.1. Stateful header fields ............................ 18
   4.2. Certificate Requirements ........................... 19
5. Transferring a signed exchange ....................... 20
   5.1. Same-origin response .............................. 20
   5.1.1. Significant headers for a same-origin response .. 20
   5.1.2. The Signed-Headers Header .................... 21
   5.2. HTTP/2 extension for cross-origin Server Push .... 22
   5.3. application/signed-exchange format ................ 22
   5.3.1. Cross-origin trust in application/signed-exchange .. 23
   5.3.2. Example ........................................ 23
6. Security considerations .................................. 23
7. Privacy considerations .................................. 23
8. IANA considerations .................................... 24
   8.1. Internet Media Type application/signed-exchange ... 24
9. References ............................................... 25
   9.1. Normative References ............................. 25
   9.2. Informative References ............................ 27
   9.3. URIs ............................................... 28
Appendix A. Change Log .................................... 28
Appendix B. Acknowledgements ............................ 30
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/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 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 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]. Its value MUST be a parameterised list (Section 3.3 of [I-D.ietf-httpbis-header-structure]), 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"  Binary content (Section 3.9 of [I-D.ietf-httpbis-header-structure]) holding the signature of most of these parameters and the exchange’s headers.
"integrity"  A string (Section 3.7 of [I-D.ietf-httpbis-header-structure]) containing the lowercase name of the response header field that guards the response payload's integrity.

"cert-url"  A string (Section 3.7 of [I-D.ietf-httpbis-header-structure]) containing an absolute URL (Section 2) with a scheme of "https" or "data".

"cert-sha256"  Binary content (Section 3.9 of [I-D.ietf-httpbis-header-structure]) holding the SHA-256 hash of the first certificate found at "cert-url".

"validity-url"  A string (Section 3.7 of [I-D.ietf-httpbis-header-structure]) containing an absolute URL (Section 2) with a scheme of "https".

"date" and "expires"  An integer (Section 3.5 of [I-D.ietf-httpbis-header-structure]) 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="MEUClQDX12gN3RNBlgFluANNFPpZCxD1aUpX6HlEwStEC0cZYLAIga9DsUQM+q5YpwEBdGW3sS+bvmM AJJ1SMWhuBdyp5UY=*
    integrity="mi-draft2";
    validity-url="https://example.com/resource.validity.1511128380";
    cert-url="https://example.com/oldcerts";
    cert-sha256="W7uB969dFW3Mb5ZefPS97q52BH51SMeOljpv2qEArmI=";
    date=1511128380; expires=1511733180

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

It relies on the "MI-Draft2" response header 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 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 (Section 2) with a scheme of "https".

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

     Note: "Host" is excluded because it is already part of the effective request URI.

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

HTTP/1.1 200  
Content-Type: text/html  
MI-Draft2: mi-sha256-draft2=dcRDgR2GM35D1uAV13PzgnG6-pvQwPwFvAulUEFrS  
Signed-Headers: "content-type", "mi-draft2"

<!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/',  
  'accept': '*/*',  
  ':method': 'GET',  
},  
{  
  'mi-draft2': 'mi-sha256-draft2=dcRDgR2GM35D1uAV13PzgnG6-pvQwPwFvAulUEFrS',  
  ':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:
Internet-Draft

Signed HTTP Exchanges Implementation Checkpoints

July 2018

cert-chain = [
  "📜⛓", ; 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 4.2.3 of [I-D.ietf-httpbis-header-structure]) 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. 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 "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 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.

3. If "integrity" names a header field other than "MI-Draft2" or this header field is not present in "exchange"’s response headers, then return "invalid". If validating integrity using the selected header field requires the client to process records larger than 16kB (for example, if the "mi-sha256-draft2" record length is greater than 16kB), return "invalid". Clients MUST be able to check the integrity of "payload" using the "MI-Draft2" header field with its "mi-sha256-draft2" content encoding, which are defined equivalently to the "MI" header field and "mi-sha256" content encoding from [I-D.thomson-http-mice].
4. 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 [I-D.ietf-tls-tls13].

      6. Otherwise, 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 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 b1".

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

         1. If "cert-sha256" is set:
1. The text string "cert-sha256" to the byte string value of "cert-sha256".

2. The text string "validity-url" to the byte string value of "validity-url".

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 "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 [I-D.ietf-tls-tls13]), in order to allow updating the stapled OCSP response without updating signatures at the same time.

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-Draft2" ([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 4) 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 ([I-D.ietf-cbor-cddl]):

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

The elements of the "signatures" array are parameterised identifiers (Section 4.2.4 of [I-D.ietf-httpbis-header-structure]) 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).
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": [ 
    {   
      "sig": "MEUCIQ....", 
      "validity-url": "https://example.com/resource.validity.1511157180"; 
      "cert-url": "https://example.com/oldcerts"; 
      "date": 1511128380; "expires": 1511733180 
    }, 
    {   
      "sig": "MEQCIC/I9Q7BZP6cSDsW43pBALSujTBpNU/+7RwKVK+bA5AlB3FSFLZgpzdJ0U6WFlV04pqgJE99fyh86UjkJkPa4jy==", 
      "validity-url": "https://example.com/resource.validity.1511157180"; 
      "cert-url": "https://example.com/newcerts"; 
      "cert-sha256": "*J/lEm9kNROdCmINbvitpvdYENQ+YgBj99D1Yp4FExw=="; 
      "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

"Signature" header fields cost on the order of 300 bytes for ECDSA signatures, so servers might prefer to avoid sending them to clients that don’t intend to use them. A client can send the "Accept-Signature" header field to indicate that it does intend to take...
advantage of any available signatures and to indicate what kinds of signatures it supports.

When a server receives an "Accept-Signature" header field in a client request, it SHOULD reply with any available "Signature" header fields for its response that the "Accept-Signature" header field indicates the client supports. However, if the "Accept-Signature" value violates a requirement in this section, the server MUST behave as if it hadn’t received any "Accept-Signature" header at all.

The "Accept-Signature" header field is a Structured Header as defined by [I-D.ietf-httpbis-header-structure]. Its value MUST be a parameterised list (Section 3.3 of [I-D.ietf-httpbis-header-structure]). Its ABNF is:

```
Accept-Signature = sh-param-list
```

The order of identifiers in the "Accept-Signature" list is not significant. Identifiers, ignoring any initial "-" character, MUST NOT be duplicated.

Each identifier in the "Accept-Signature" header field’s value indicates that a feature of the "Signature" header field (Section 3.1) is supported. If the identifier begins with a "-" character, it instead indicates that the feature named by the rest of the identifier is not supported. Unknown identifiers and parameters MUST be ignored because new identifiers and new parameters on existing identifiers may be defined by future specifications.

### 3.7.1. Integrity identifiers

Identifiers starting with "mi-draft2/" indicate that the client supports the "MI-Draft2" header field (equivalent to "MI" in [I-D.thomson-http-mice]) with the parameter from the HTTP MI Parameter Registry registry named in lower-case by the rest of the identifier. For example, "mi-draft2/mi-blake2" indicates support for Merkle integrity with the as-yet-unspecified mi-blake2 parameter, and "-mi-draft2/mi-sha256-draft2" indicates non-support for Merkle integrity with the mi-sha256-draft2 content encoding.

If the "Accept-Signature" header field is present, servers SHOULD assume support for "mi-draft2/mi-sha256-draft2" unless the header field states otherwise.
3.7.2. Key type identifiers

Identifiers starting with "ecdsa/" indicate that the client supports certificates holding ECDSA public keys on the curve named in lowercase by the rest of the identifier.

If the "Accept-Signature" header field is present, servers SHOULD assume support for "ecdsa/secp256r1" unless the header field states otherwise.

3.7.3. Key value identifiers

The "ed25519key" identifier has parameters indicating the public keys that will be used to validate the returned signature. Each parameter's name is re-interpreted as binary content (Section 3.9 of [I-D.ietf-httpbis-header-structure]) encoding a prefix of the public key. For example, if the client will validate signatures using the public key whose base64 encoding is "11qYAYKxCrfVS/7TyWQHoG7hcvPapiM1rw1aaPcHUrO==", valid "Accept-Signature" header fields include:

Accept-Signature: ..., ed25519key; *11qYAYKxCrfVS/7TyWQHoG7hcvPapiM1rw1aaPcHUrO==*
Accept-Signature: ..., ed25519key; *11qYAYKxCrfVS/7TyWQHoG==*
Accept-Signature: ..., ed25519key; *11qYA==*
Accept-Signature: ..., ed25519key; **

but not

Accept-Signature: ..., ed25519key; *11qYA===*

because 5 bytes isn't a valid length for encoded base64, and not

Accept-Signature: ..., ed25519key; 11qYAQ

because it doesn't start or end with the "**"s that indicate binary content.

Note that "ed25519key; **" is an empty prefix, which matches all public keys, so it's useful in subresource integrity cases like "<link rel=preload as=script href="...">" where the public key isn't known until the matching "<script src="..." integrity="...">" tag.

3.7.4. Examples

Accept-Signature: mi-draft2/mi-sha256

states that the client will accept signatures with payload integrity assured by the "MI-Draft2" header and "mi-sha256-draft2" content
encoding and implies that the client will accept integrity assured by the "Digest: SHA-256" header and signatures from ECDSA keys on the secp256r1 curve.

Accept-Signature: -ecdsa/secp256r1, ecdsa/secp384r1

states that the client will accept ECDSA keys on the secp384r1 curve but not the secp256r1 curve and payload integrity assured with the "MI-Draft2: mi-sha256-draft2" header field.

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 "validity-url" parameter (Paragraph 5) is not same-origin [6] 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-
2. Validate that "main-certificate" has the CanSignHttpExchanges extension (Section 4.2).

3. 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.

4. 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].

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]
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

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.

Conforming CAs MUST NOT mark this extension as critical.

Clients MUST NOT accept certificates with this extension in TLS connections (Section 4.4.2.2 of [I-D.ietf-tls-tls13]).

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.

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

The signature for a signed exchange can be included in a normal HTTP response. Because different clients send different request header fields, and intermediate servers add response header fields, it can be impossible to have a signature for the exact request and response that the client sees. Therefore, when a client validates the "Signature" header field for an exchange represented as a normal HTTP request/response pair, it MUST pass only the subset of header fields defined by Section 5.1.1 to the validation procedure (Section 3.5).

If the client relies on signature validity for any aspect of its behavior, it MUST ignore any header fields that it didn’t pass to the validation procedure.

5.1.1. Significant headers for a same-origin response

The significant headers of an exchange represented as a normal HTTP request/response pair (Section 2.1 of [RFC7230] or Section 8.1 of [RFC7540]) are:

- The method (Section 4 of [RFC7231]) and effective request URI (Section 5.5 of [RFC7230]) of the request.
The response status code (Section 6 of [RFC7231]) and the response header fields whose names are listed in that exchange’s "Signed-Headers" header field (Section 5.1.2), in the order they appear in that header field. If a response header field name from "Signed-Headers" does not appear in the exchange’s response header fields, the exchange has no significant headers.

If the exchange’s "Signed-Headers" header field is not present, doesn’t parse as a Structured Header ([I-D.ietf-httpbis-header-structure]) or doesn’t follow the constraints on its value described in Section 5.1.2, the exchange has no significant headers.

5.1.2. The Signed-Headers Header

The "Signed-Headers" header field identifies an ordered list of response header fields to include in a signature. The request URL and response status are included unconditionally. This allows a TLS-terminating intermediate to reorder headers without breaking the signature. This _can_ also allow the intermediate to add headers that will be ignored by some higher-level protocols, but Section 3.5 provides a hook to let other higher-level protocols reject such insecure headers.

This header field appears once instead of being incorporated into the signatures’ parameters because the signed header fields need to be consistent across all signatures of an exchange, to avoid forcing higher-level protocols to merge the header field lists of valid signatures.

"Signed-Headers" is a Structured Header as defined by [I-D.ietf-httpbis-header-structure]. Its value MUST be a list (Section 3.2 of [I-D.ietf-httpbis-header-structure]). Its ABNF is:

Signed-Headers = sh-list

Each element of the "Signed-Headers" list must be a lowercase string (Section 3.7 of [I-D.ietf-httpbis-header-structure]) naming an HTTP response header field. Pseudo-header field names (Section 8.1.2.1 of [RFC7540]) MUST NOT appear in this list.

Higher-level protocols SHOULD place requirements on the minimum set of headers to include in the "Signed-Headers" header field.
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 request metadata and header fields, response metadata and header fields, and a response payload.

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

1. The ASCII characters "sxg1-b1" 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 stop parsing if they don’t.

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

2. 3 bytes storing a big-endian integer "sigLength". If this is larger than 16kB, parsing MUST fail.

3. 3 bytes storing a big-endian integer "headerLength". If this is larger than 512kB, parsing MUST fail.

4. "sigLength" bytes holding the "Signature" header field’s value (Section 3.1).

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

   Note that this is exactly the bytes used when checking signature validity in Section 3.5.

6. 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 and an exchange consisting of the headers from the signed headers section and the payload body, to the algorithm in Section 4.

5.3.2. Example

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

sxg1-b1\0<3-byte length of the following header
value><3-byte length of the encoding of the
following array>sig1; sig=*...; integrity="mi-draft2"; ...[

{ 
  ':method': 'GET',
  ':url': 'https://example.com/',
  'accept', '*/*'
},

{ 
  ':status': '200',
  'content-type': 'text/html'
}
]
<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 ([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 registrations in [I-D.yasskin-http-origin-signed-responses]:

- The "Signature" header field
- The "Accept-Signature" header field
- The "Signed-Headers" 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 "b1". 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 31 00

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

9. References

9.1. Normative References


9.2. Informative References

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

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


9.3. URIs

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

Appendix A. Change Log

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 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