Signed HTTP Exchanges
draft-yasskin-http-origin-signed-responses-02

Abstract

This document specifies how a server can send an HTTP request/response pair, known as an exchange, with signatures that vouch for that exchange’s authenticity. These signatures can be verified against an origin’s certificate to establish that the exchange is authoritative for an origin even if it was transferred over a connection that isn’t. The signatures can also be used in other ways described in the appendices.

These signatures contain countermeasures against downgrade and protocol-confusion attacks.

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 July 30, 2018.
Table of Contents

1. Introduction ....................................................... 3
2. Terminology ....................................................... 4
3. Signing an exchange .............................................. 5
   3.1. The Signed-Headers Header .................................. 5
   3.2. The Signature Header ........................................ 6
       3.2.1. Examples ............................................... 7
       3.2.2. Open Questions ....................................... 8
   3.3. Significant headers of an exchange ............................ 9
       3.3.1. Open Questions ....................................... 9
   3.4. CBOR representation of exchange headers ....................... 9
       3.4.1. Example ............................................... 10
   3.5. Canonical CBOR serialization ................................ 10
   3.6. Signature validity .......................................... 11
       3.6.1. Open Questions ....................................... 15
   3.7. Updating signature validity .................................. 15
       3.7.1. Examples ............................................... 16
   3.8. The Accept-Signature header .................................. 18
       3.8.1. Integrity labels ...................................... 19
       3.8.2. Key type labels ...................................... 19
       3.8.3. Key value labels .................................... 19
       3.8.4. Examples ............................................... 20
       3.8.5. Open Questions ....................................... 20
4. HTTP/2 extension for cross-origin Server Push .................... 21
   4.1. Indicating support for cross-origin Server Push ............... 21
   4.2. NO_TRUSTED.Exchange_SIGNATURE error code .................. 21
       4.2.1. Open Questions ....................................... 22
   4.3. Validating a cross-origin Push ................................ 22
       4.3.1. Validating a certificate chain for an authority ....... 22
       4.3.2. Open Questions ....................................... 23
5. application/http-exchange+cbor format for HTTP/1 compatibility .... 23
1. Introduction

Signed HTTP exchanges provide a way to prove the authenticity of a resource in cases where the transport layer isn’t sufficient. This can be used in several ways:
o When signed by a certificate ([RFC5280]) that’s trusted for an origin, an exchange can be treated as authoritative for that origin, even if it was transferred over a connection that isn’t authoritative (Section 9.1 of [RFC7230]) for that origin. See Appendix A.1 and Appendix A.2.

o A top-level resource can use a public key to identify an expected author for particular subresources, a system known as Subresource Integrity ([SRI]). An exchange’s signature provides the matching proof of authorship. See Appendix A.3.

o A signature can vouch for the exchange in some way, for example that it appears in a transparency log or that static analysis indicates that it omits certain attacks. See Appendix A.4 and Appendix A.5.

Subsequent work toward the use cases in [I-D.yasskin-webpackage-use-cases] will provide a way to group signed exchanges into bundles that can be transmitted and stored together, but single signed exchanges are useful enough to standardize on their own.

2. Terminology

Author  The entity that controls the server for a particular origin [RFC6454]. The author 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 author 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 author 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

As a response to an HTTP request or as a Server Push (Section 8.2 of [RFC7540]) the server MAY include a "Signed-Headers" header field (Section 3.1) identifying significant (Section 3.3) header fields and a "Signature" header field (Section 3.2) holding a list of one or more parameterised signatures that vouch for the content of the response.

The client categorizes each signature as "valid" or "invalid" by validating that signature with its certificate or public key and other metadata against the significant headers and content (Section 3.6). 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 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 3.6 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 significant 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.

See Appendix B.2 for a discussion of why only the URL from the request is included and not other request headers.

"Signed-Headers" is a Structured Header as defined by [I-D.ietf-httpbis-header-structure]. Its value MUST be a list (Section 4.8 of [I-D.ietf-httpbis-header-structure]) of lowercase strings (Section 4.2 of [I-D.ietf-httpbis-header-structure]) naming HTTP response header fields. 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.

3.2. The Signature Header

The "Signature" header field conveys a list of signatures for an exchange, each one accompanied by information about how to determine the authority of and refresh that signature. Each signature directly signs the significant headers of the exchange 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 list (Section 4.8 of [I-D.ietf-httpbis-header-structure]) of parameterised labels (Section 4.4 of [I-D.ietf-httpbis-header-structure]).

Each parameterised label MUST have parameters named "sig", "integrity", "validityUrl", "date", and "expires". Each parameterised label MUST also have either "certUrl" and "certSha256" parameters or an "ed25519Key" parameter. This specification gives no meaning to the label itself, which can be used as a human-readable identifier for the signature (see Section 3.2.2, Paragraph 1). The present parameters MUST have the following values:

"sig" Binary content (Section 4.5 of [I-D.ietf-httpbis-header-structure]) holding the signature of most of these parameters and the significant headers of the exchange (Section 3.3).

"integrity" A string (Section 4.2 of [I-D.ietf-httpbis-header-structure]) 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]) containing a valid URL string [4].

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

"ed25519Key" Binary content (Section 4.5 of [I-D.ietf-httpbis-header-structure]) holding an Ed25519 public key ([RFC8032]).
"validityUrl" A string (Section 4.2 of [I-D.ietf-httpbis-header-structure]) containing a valid URL string [5].

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

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

3.2.1. 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="MEUCIQDXlI2gN3RNBlgfUuRNFn2Xc1aUpX6H1Ewc3Ec0c2YLA1Iga9SaVOMM+g5YpwEBdGW3sS+bvnmAJJjISwhuBdgp5UY;
integrity="mi";
validityUrl=https://example.com/resource.validity.1511128380";
certUrl=https://example.com/oldcerts";
certSha256=*W7uB969dFWJMb5eEFPS9Tq5ZbH5lsMoIL3jv2qEArmI;
date=15111228380; expires=1511733180,
sig2;
sig="MEQCIGjZRgTRf9iKNkGFyzRMTFgwf/BryZ2N1p/dykhuVUoYaAI8BTyk+8wujoT4n/M+cNg7pGqV1UGYE8u8HZJ5Yh26Q9;
integrity="mi";
validityUrl=https://example.com/resource.validity.1511128380";
certUrl=https://example.com/newcerts";
certSha256=+71J0m9kHRODdCM71nbvtpvdYKNq+yGbJ95DlyP4fKxw;
date=15111228380; expires=1511733180,
srsg;
sig="1GzV6aJMJ5f2oGoGzF1mBdKTDL+QADza48Ge04944gACYJOvzof6uhOJCcwKrK7DK+LBCh0jvDYPfP5CLc1SDA
integrity="mi";
validityUrl=https://example.com/resource.validity.1511128380";
ed25519Key=*zsSevyFsxyZMiUIuVBD4eydpRLTgynW0VouKu+gA8
date=15111228380; expires=1511733180,
thirdparty;
sig="MEYCIGQwnJzn6H2f2Nssobktirr7KkajYQFmWUtW114PewQaHMs2TVjc4r7shD7xtXbqEQoWqj2mRXALhFZxFzXgPupi+
integrity="mi";
validityUrl=https://thirdparty.example.com/resource.validity.1511161860";
certUrl=https://thirdparty.example.com/certs";
certSha256=O6xOzU2kX1GTyVvHcsmWNGA20NaZdUBdEvgRA9ZanNc;
date=1511133060; expires=1511478660,

There are 4 signatures: 2 from different secp256r1 certificates within "https://example.com/", one using a raw ed25519 public key
that’s also controlled by "example.com", and a fourth using a secp256r1 certificate owned by "thirdparty.example.com".

All 4 signatures rely on the "MI" response header to guard the integrity of the response payload. This isn’t strictly required—some signatures could use "MI" while others use "Digest"—but there’s not much benefit to mixing them.

The signatures include 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 certificates at "https://example.com/oldcerts" and "https://example.com/newcerts" have "subjectAltName"s of "example.com", meaning that if they and their signatures validate, the exchange can be trusted as having an origin of "https://example.com/". The author might be using two certificates because their readers have disjoint sets of roots in their trust stores.

The author signed with all three certificates at the same time, so they share a validity range: 7 days starting at 2017-11-19 21:53 UTC.

The author then requested an additional signature from "thirdparty.example.com", which did some validation or processing and then signed the resource at 2017-11-19 23:11 UTC. "thirdparty.example.com" only grants 4-day signatures, so clients will need to re-validate more often.

3.2.2. Open Questions

[I-D.ietf-httpbis-header-structure] 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] allows, so they’re single items for now.
3.3. Significant headers of an exchange

The significant headers of an exchange 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 3.1), 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 3.1, the exchange has no significant headers.

3.3.1. Open Questions

Do the significant headers of an exchange need to include the "Signed-Headers" header field itself?

3.4. CBOR representation of exchange headers

To sign an exchange’s headers, they need to be serialized into a byte string. Since intermediaries and distributors (Appendix A.2) 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 3.5), 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.

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 name as a byte string to the header field’s value as a
  byte string.

3.4.1. Example

Given the HTTP exchange:

GET https://example.com/ HTTP/1.1
Accept: */*

HTTP/1.1 200
Content-Type: text/html
Digest: SHA-256=20addcf7368837f616d549f035bf6784ea6d4bf4817a3736cd2fc7a763897fe3
Signed-Headers: "content-type", "digest"

<!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',
  },  
  {   
    'digest': 'SHA-256=20addcf7368837f616d549f035bf6784ea6d4bf4817a3736cd2fc7a763897fe3',
    ':status': '200',
    'content-type': 'text/html'
  }  
]

3.5. 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:

  o Integers and the lengths of arrays, maps, and strings MUST use the
    smallest possible encoding.
o Items MUST NOT be encoded with indefinite length.

o 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.

3.6. 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]) described in Section 3.2. 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. Potentially-valid results include:

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

o 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.

This algorithm also accepts an "allResponseHeaders" flag, which
insists that there are no non-significant response header fields in
the exchange.

1. Let "originalExchange" be the signature’s exchange.

2. Let "headers" be the significant headers (Section 3.3) of
"originalExchange". If "originalExchange" has no significant
headers, then return "invalid".

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

4. If "allResponseHeaders" is set and the response header fields in
"originalExchange" are not equal to the response header fields
in "headers", then return "invalid".

5. 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.
* "ed25519Key" be the signature’s "ed25519Key" 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.

6. If "integrity" names a header field that is not present in
"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 implement at least the "Digest" ([RFC3230]) and "MI" ([I-D.thomson-http-mice]) header fields.

7. If "integrity" is "digest", and the "Digest" header field in "headers" contains no digest-algorithms (https://www.iana.org/assignments/http-dig-alg/http-dig-alg.xhtml [6]) stronger than "SHA", then return "invalid".

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

   1. If "certUrl" is present:
      1. Let "certificate-chain" be the result of fetching ([FETCH]) "certUrl" and parsing it as a TLS 1.3 Certificate message (Section 4.4.2 of [I-D.ietf-tls-tls13]) containing X.509v3 certificates. If "forceFetch" is _not_ set, the fetch can be fulfilled from a cache using normal HTTP semantics [RFC7234]. If this fetch or parse fails, return "invalid".

      Parsing notes: 1. This does not include the 4-byte header that would appear in a Handshake message. 1. 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.

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

      3. If the SHA-256 hash of "main-certificate"’s "cert_data" is not equal to "certSha256", 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.

      4. Set "publicKey" to "main-certificate"’s public key

      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_sha256 as defined in Section 4.2.3 of [I-D.ietf-tls-tls13].
EC, with the secp256r1 curve: ecdsa_secp256r1_sha256 as defined in Section 4.2.3 of [I-D.ietf-tls-tls13].

EC, with the secp384r1 curve: ecdsa_secp384r1_sha384 as defined in Section 4.2.3 of [I-D.ietf-tls-tls13].

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

2. If "ed25519Key" is present, set "publicKey" to "ed25519Key" and "signing-alg" to ed25519, as defined by [RFC8032]

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

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

11. 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 3.5) 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 3.4) of "exchange"'s headers.

12. If "signature" is "message"'s signature by "main-certificate"'s public key using "signing-alg", return "potentially-valid" with "exchange" and whichever is present of "certificate-chain" or "ed25519Key". 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.

3.6.1. Open Questions

Should we ban RSA keys to avoid their vulnerability to Bleichenbacher attacks?

3.7. 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 "certUrl" (Section 3.6, Paragraph 4) to get a chain including a newer OCSP response.

The "validityUrl" parameter (Paragraph 6) 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]) 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.

3.7.1.  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,
sig2;
sig=*MEQCIG...;
...validityUrl="https://example.com/resource.validity.1511157180";
certUrl="https://example.com/newcerts";
date=1511128380; expires=1511733180,
thirdpartysig;
sig=*MEYCIQ...;
...validityUrl="https://thirdparty.example.com/resource.validity.1511161860";
certUrl="https://thirdparty.example.com/certs";
date=1511478660; expires=1511824260

At 2017-11-27 11:02 UTC, "sig1" and "sig2" have expired, but
"thirdpartysig" doesn’t expire until 23:11 that night, so the client
needs to fetch "https://example.com/resource.validity.1511157180"
(the "validityUrl" of "sig1" and "sig2") to update those signatures.
This URL might contain:

```
{  
  "signatures": [  
    {"sig1": "MEU6IC...;
    "sig=*MEQCIC/I9Q+7BZFP6cSDsWx43pBAl0uj7bDN+/7rKXvbaA5AL3/3FSLdvgzmDJO5umWwNw4pqg3W99fcK/W6Uj3/fh4jw;  
    "validityUrl="https://example.com/resource.validity.1511157180";  
    "integrity="mi;  
    "certUrl="https://example.com/newcerts";  
    "certSha256=*J/lEm9kNROOdCmJ2vqVpVYjKMQY+YqBjJ9D1Yp4fEw;  
    "date=15111733180; 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 first two of the original signatures (the ones with a
validityUrl of "https://example.com/resource.validity.1511157180")
with the single new signature provided. (This might happen at the
end of a migration to a new root certificate.) The signatures 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,
thirdpartysig;
sig=*MYCIQ...;
...
validityUrl="https://thirdparty.example.com/resource.validity.1511161860";
certUrl="https://thirdparty.example.com/certs";
date=1511478660; expires=1511824260

"https://example.com/resource.validity.1511157180" could also expand the set of signatures if its "signatures" array contained more than 2 elements.

3.8. 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 list (Section 4.8 of [I-D.ietf-httpbis-header-structure]) of parameterised labels (Section 4.4 of [I-D.ietf-httpbis-header-structure]). The order of labels in the "Accept-Signature" list is not significant. Labels, ignoring any initial "-" character, MUST NOT be duplicated.

Each label in the "Accept-Signature" header field’s value indicates that a feature of the "Signature" header field (Section 3.2) is supported. If the label begins with a "-" character, it instead indicates that the feature named by the rest of the label is not
supported. Unknown labels and parameters MUST be ignored because new labels and new parameters on existing labels may be defined by future specifications.

3.8.1. Integrity labels

Labels starting with "digest/" indicate that the client supports the "Digest" header field ([RFC3230]) with the digest-algorithm from the https://www.iana.org/assignments/http-dig-alg/http-dig-alg.xhtml [7] registry named in lower-case by the rest of the label. For example, "digest/sha-512" indicates support for the SHA-512 digest algorithm, and "-digest/sha-256" indicates non-support for the SHA-256 digest algorithm.

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

If the "Accept-Signature" header field is present, servers SHOULD assume support for "digest/sha-256" and "mi/mi-sha256" unless the header field states otherwise.

3.8.2. Key type labels

Labels starting with "rsa/" indicate that the client supports certificates holding RSA public keys with a number of bits indicated by the digits after the "/".

Labels starting with "ecdsa/" indicate that the client supports certificates holding ECDSA public keys on the curve named in lower-case by the rest of the label.

If the "Accept-Signature" header field is present, servers SHOULD assume support for "rsa/2048", "ecdsa/secp256r1", and "ecdsa/secp384r1" unless the header field states otherwise.

3.8.3. Key value labels

The "ed25519key" label 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 4.5 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/7TyWQHOg7hcVPapiMlrwIaaPcHURO", valid "Accept-Signature" header fields include:

Accept-Signature: ..., ed25519key; *11qYAYKxCrfVS/7TyWQHOg7hcVPapiMlrwIaaPcHURO
Accept-Signature: ..., ed25519key; *11qYAYKxCrfVS/7TyWQHOg
Accept-Signature: ..., ed25519key; *11qYAQ
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 with the "*" that indicates binary content.

Note that "ed25519key; *" is an empty prefix, which matches all public keys, so it’s useful in subresource integrity (Appendix A.3) cases like "<link rel=preload as=script href="...">" where the public key isn’t known until the matching "<script src="..." integrity="...">" tag.

3.8.4. Examples

Accept-Signature: mi/mi-sha256

states that the client will accept signatures with payload integrity assured by the "MI" header and "mi-sha256" content encoding and implies that the client will accept integrity assured by the "Digest: SHA-256" header and signatures from 2048-bit RSA keys and ECDSA keys on the secp256r1 and secp384r1 curves.

Accept-Signature: -rsa/2048, rsa/4096

states that the client will accept 4096-bit RSA keys but not 2048-bit RSA keys, and implies that the client will accept ECDSA keys on the secp256r1 and secp384r1 curves and payload integrity assured with the "MI: mi-sha256" and "Digest: SHA-256" header fields.

3.8.5. Open Questions

Is an "Accept-Signature" header useful enough to pay for itself? If clients wind up sending it on most requests, that may cost more than the cost of sending "Signature"s unconditionally. On the other hand, it gives servers an indication of which kinds of signatures are supported, which can help us upgrade the ecosystem in the future.
Is "Accept-Signature" the right spelling, or do we want to imitate "Want-Digest" (Section 4.3.1 of [RFC3230]) instead?

Do I have the right structure for the labels indicating feature support?

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

To allow servers to Server-Push (Section 8.2 of [RFC7540]) signed exchanges (Section 3) signed by an authority for which the server is not authoritative (Section 9.1 of [RFC7230]), this section defines an HTTP/2 extension.

4.1. Indicating support for cross-origin Server Push

Clients that might accept signed Server Pushes with an authority for which the server is not authoritative indicate this using the HTTP/2 SETTINGS parameter ENABLE_CROSS_ORIGIN_PUSH (0xSETTING-TBD).

An ENABLE_CROSS_ORIGIN_PUSH value of 0 indicates that the client does not support cross-origin Push. A value of 1 indicates that the client does support cross-origin Push.

A client MUST NOT send a ENABLE_CROSS_ORIGIN_PUSH setting with a value other than 0 or 1 or a value of 0 after previously sending a value of 1. If a server receives a value that violates these rules, it MUST treat it as a connection error (Section 5.4.1 of [RFC7540]) of type PROTOCOL_ERROR.

The use of a SETTINGS parameter to opt-in to an otherwise incompatible protocol change is a use of "Extending HTTP/2" defined by Section 5.5 of [RFC7540]. If a server were to send a cross-origin Push without first receiving a ENABLE_CROSS_ORIGIN_PUSH setting with the value of 1 it would be a protocol violation.

4.2. NO_TRUSTED_EXCHANGE_SIGNATURE error code

The signatures on a Pushed cross-origin exchange may be untrusted for several reasons, for example that the certificate could not be fetched, that the certificate does not chain to a trusted root, that the signature itself doesn’t validate, that the signature is expired, etc. This draft conflates all of these possible failures into one error code, NO_TRUSTED_EXCHANGE_SIGNATURE (0xERROR-TBD).
4.2.1. Open Questions

How fine-grained should this specification’s error codes be?

4.3. Validating a cross-origin Push

If the client has set the ENABLE_CROSS_ORIGIN_PUSH setting to 1, the server MAY Push a signed exchange for which it is not authoritative, and the client MUST NOT treat a PUSH_PROMISE for which the server is not authoritative as a stream error (Section 5.4.2 of [RFC7540]) of type PROTOCOL_ERROR, as described in Section 8.2 of [RFC7540].

Instead, the client MUST validate such a PUSH_PROMISE and its response by parsing the "Signature" header into a list of signatures according to the instructions in Section 3.6, and searching that list for a valid signature using the algorithm in Section 4.3.1. If no valid signature is found, the client MUST treat the response as a stream error (Section 5.4.2 of [RFC7540]) of type NO_TRUSTED_EXCHANGE_SIGNATURE. Otherwise, the client MUST treat the pushed response as if the server were authoritative for the PUSH_PROMISE’s authority.

4.3.1. Validating a certificate chain for an authority

1. If the signature’s "validityUrl" parameter (Paragraph 6) is not same-origin [8] with the exchange’s effective request URI (Section 5.5 of [RFC7230]), return "invalid".

2. Run Section 3.6 over the signature with the "allResponseHeaders" flag set, getting "exchange" and "certificate-chain" back. If this returned "invalid" or didn’t return a certificate chain, return "invalid".

3. Let "authority" be the host component of "exchange"’s effective request URI.

4. Validate the "certificate-chain" using the following substeps. If any of them fail, re-run Section 3.6 once over the signature with both the "forceFetch" flag and the "allResponseHeaders" 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" includes a "status_request" extension 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.

3. Validate that all certificates in "path" include "signed_certificate_timestamp" extensions containing valid SCTs from trusted logs. ([RFC6962])

5. Return "valid".

4.3.2. Open Questions

Is it right that "validityUrl" is required to be same-origin with the exchange? This allows the mitigation against downgrades in Section 6.3, but prohibits intermediates from providing a cache of the validity information. We could do both with a list of URLs.

5. application/http-exchange+cbor format for HTTP/1 compatibility

To allow servers to serve cross-origin responses when either the client or the server hasn't implemented HTTP/2 Push (Section 8.2 of [RFC7540]) support yet, we define a format that represents an HTTP exchange.

The "application/http-exchange+cbor" content type encodes an HTTP exchange, including request metadata and header fields, optionally a request body, response header fields and metadata, a payload body, and optionally trailer header fields.

This content type consists of a canonically-serialized (Section 3.5) CBOR array containing:

1. The text string "htxg" to serve as a file signature, followed by
2. Alternating member names encoded as text strings (Section 2.1 of [RFC7049]) and member values, with each value consisting of a single CBOR item with a type and meaning determined by the member name.

This specification defines the following member names with their associated values:

"request" A map from request header field names to values, encoded as byte strings ([RFC7049], section 2.1). The request header
fields MUST include two pseudo-header fields (Section 8.1.2.1 of [RFC7540]):

* "':method'": The method of the request (Section 4 of [RFC7231]).

* "':url'": The effective request URI of the request (Section 5.5 of [RFC7230]).

"request payload" A byte string ([RFC7049], section 2.1) containing the request payload body (Section 3.3 of [RFC7230]).

"response" A map from response header field names to values, encoded as byte strings ([RFC7049], section 2.1). The response header fields MUST include one pseudo-header field (Section 8.1.2.1 of [RFC7540]):

* "':status'": The response’s 3-digit status code (Section 6 of [RFC7231]).

"payload" A byte string ([RFC7049], section 2.1) containing the response payload body (Section 3.3 of [RFC7230]).

"trailer" A map of trailer header field names to values, encoded as byte strings (Section 2.1 of [RFC7049]).

A parser MAY return incremental information while parsing "application/http-exchange+cbor" content.

Members "request", "response", and "payload" MUST be present. If one is missing, the parser MUST stop and report an error.

The member names MUST appear in the order:

1. "request"
2. "request payload"
3. "response"
4. "payload"
5. "trailer"

If a member name is not a text string, appears out of order, or is followed by a value not matching its description above, the parser MUST stop and report an error.
If the parser encounters an unknown member name, it MUST skip the following item and resume parsing at the next member name.

5.1. Example

An example "application/http-exchange+cbor" file representing a possible exchange with https://example.com/ [9] follows, in the extended diagnostic format defined in Appendix G of [I-D.ietf-cbor-cddl]:

```
[ 
  "htxg",
  "request",
  {
    ':method': 'GET',
    ':url': 'https://example.com/',
    'accept', '*/*'
  },
  "response",
  {
    ':status': '200',
    'content-type': 'text/html'
  },
  "payload",
  '<!doctype html>
  \n  <html>...
```

5.2. Open Questions

Should "application/http-exchange+cbor" support request payloads and trailers, or only the aspects needed for signed exchanges?

Are the mime type, extension, and magic number right?

6. Security considerations

6.1. Confidential data

Authors MUST NOT include confidential information in a signed response that an untrusted intermediate could forward, since the response is only signed and not encrypted. Intermediates can read the content.

6.2. Off-path attackers

Relaxing the requirement to consult DNS when determining authority for an origin means that an attacker who possesses a valid certificate no longer needs to be on-path to redirect traffic to
them; instead of modifying DNS, they need only convince the user to visit another Web site in order to serve responses signed as the target. This consideration and mitigations for it are shared by the combination of [I-D.ietf-httpbis-origin-frame] and [I-D.ietf-httpbis-http2-secondary-certs].

6.3. Downgrades

Signing a bad response can affect more users than simply serving a bad response, since a served response will only affect users who make a request while the bad version is live, while an attacker can forward a signed response until its signature expires. Authors should consider shorter signature expiration times than they use for cache expiration times.

Clients MAY also check the "validityUrl" (Paragraph 6) of an exchange more often than the signature’s expiration would require. Doing so for an exchange with an HTTPS request URI provides a TLS guarantee that the exchange isn’t out of date (as long as Section 4.3.2 is resolved to keep the same-origin requirement).

6.4. Signing oracles are permanent

An attacker with temporary access to a signing oracle can sign "still valid" assertions with arbitrary timestamps and expiration times. As a result, when a signing oracle is removed, the keys it provided access to SHOULD be revoked so that, even if the attacker used them to sign future-dated exchange validity assertions, the key’s OCSP assertion will expire, causing the exchange as a whole to become untrusted.

6.5. Unsigned headers

The use of a single "Signed-Headers" header field prevents us from signing aspects of the request other than its effective request URI (Section 5.5 of [RFC7230]). For example, if an author signs both "Content-Encoding: br" and "Content-Encoding: gzip" variants of a response, what’s the impact if an attacker serves the brotli one for a request with "Accept-Encoding: gzip"?

The simple form of "Signed-Headers" also prevents us from signing less than the full request URL. The SRI use case (Appendix A.3) may benefit from being able to leave the authority less constrained.

Section 3.6 can succeed when some delivered headers aren’t included in the signed set. This accommodates current TLS-terminating intermediates and may be useful for SRI (Appendix A.3), but is risky for trusting cross-origin responses (Appendix A.1, Appendix A.2, and
Appendix A.6). Section 4 requires all headers to be included in the signature before trusting cross-origin pushed resources, at Ryan Sleevi’s recommendation.

6.6. application/http-exchange+cbor

Clients MUST NOT trust an effective request URI claimed by an "application/http-exchange+cbor" resource (Section 5) without either ensuring the resource was transferred from a server that was authoritative (Section 9.1 of [RFC7230]) for that URI’s origin, or validating the resource’s signature using a procedure like the one described in Section 4.3.1.

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 author.

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

TODO: possibly register the validityUrl format.
8.1. Signature Header Field Registration

This section registers the "Signature" header field in the "Permanent Message Header Field Names" registry ([RFC3864]).

Header field name: "Signature"

Applicable protocol: http

Status: standard

Author/Change controller: IETF

Specification document(s): Section 3.2 of this document

8.2. HTTP/2 Settings

This section establishes an entry for the HTTP/2 Settings Registry that was established by Section 11.3 of [RFC7540]

Name: ENABLE_CROSS_ORIGIN_PUSH

Code: 0xSETTING-TBD

Initial Value: 0

Specification: This document

8.3. HTTP/2 Error code

This section establishes an entry for the HTTP/2 Error Code Registry that was established by Section 11.4 of [RFC7540]

Name: NO_TRUSTED_EXCHANGE_SIGNATURE

Code: 0xERROR-TBD

Description: The client does not trust the signature for a cross-origin Pushed signed exchange.

Specification: This document

8.4. Internet Media Type application/http-exchange+cbor

Type name: application

Subtype name: http-exchange+cbor
Required parameters: N/A

Optional parameters: N/A

Encoding considerations: binary

Security considerations: see Section 6.6

Interoperability considerations: N/A

Published specification: This specification (see Section 5).

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): 8? 64 68 74 78 67

File extension(s): .htxg

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


[I-D.ietf-cbor-cddl]

[I-D.ietf-httpbis-header-structure]

[I-D.ietf-tls-tls13]

[I-D.thomson-http-mice]


9.2. Informative References

[I-D.burke-content-signature]
Burke, B., "HTTP Header for digital signatures", draft-burke-content-signature-00 (work in progress), March 2011.
[I-D.cavage-http-signatures]

[I-D.ietf-httpbis-http2-secondary-certs]


[I-D.thomson-http-content-signature]

[I-D.yasskin-webpackage-use-cases]
Yasskin, J., "Use Cases and Requirements for Web Packages", draft-yasskin-webpackage-use-cases-00 (work in progress), August 2017.


9.3. URIs

[1] https://lists.w3.org/Archives/Public/ietf-http-wg/
[8] https://html.spec.whatwg.org/multipage/origin.html#same-origin
[9] https://example.com/
[17] https://www.imperialviolet.org/2012/02/05/crlsets.html

Appendix A. Use cases

A.1. PUSHed subresources

To reduce round trips, a server might use HTTP/2 Push (Section 8.2 of [RFC7540]) to inject a subresource from another server into the client’s cache. If anything about the subresource is expired or
can’t be verified, the client would fetch it from the original server.

For example, if "https://example.com/index.html" includes

<script src="https://jquery.com/jquery-1.2.3.min.js">

Then to avoid the need to look up and connect to "jquery.com" in the critical path, "example.com" might push that resource signed by "jquery.com".

A.2. Explicit use of a content distributor for subresources

In order to speed up loading but still maintain control over its content, an HTML page in a particular origin "O.com" could tell clients to load its subresources from an intermediate content distributor that’s not authoritative, but require that those resources be signed by "O.com" so that the distributor couldn’t modify the resources. This is more constrained than the common CDN case where "O.com" has a CNAME granting the CDN the right to serve arbitrary content as "O.com".

<img log_seriesrc="https://O.com/img.png"
physicalsrc="https://distributor.com/O.com/img.png">

To make it easier to configure the right distributor for a given request, computation of the "physicalsrc" could be encapsulated in a custom element:

<dist-img src="https://O.com/img.png"></dist-img>

where the "<dist-img>" implementation generates an appropriate "<img>" based on, for example, a "<meta name="dist-base">" tag elsewhere in the page. However, this has the downside that the preloader [10] can no longer see the physical source to download it. The resulting delay might cancel out the benefit of using a distributor.

This could be used for some of the same purposes as SRI (Appendix A.3).

To implement this with the current proposal, the distributor would respond to the physical request to "https://distributor.com/O.com/img.png" with first a signed PUSH_PROMISE for "https://O.com/img.png" and then a redirect to "https://O.com/img.png".
A.3. Subresource Integrity

The W3C WebAppSec group is investigating using signatures [11] in [SRI]. They need a way to transmit the signature with the response, which this proposal provides.

Their needs are simpler than most other use cases in that the "integrity="ed25519-\{public-key\}" attribute and CSP-based ways of expressing a public key don’t need that key to be wrapped into a certificate.

The "ed25519Key" signature parameter supports this simpler way of attaching a key.

The current proposal for signature-based SRI describes signing only the content of a resource, while this specification requires them to sign the request URI as well. This issue is tracked in https://github.com/mikewest/signature-based-sri/issues/5 [12]. The details of what they need to sign will affect whether and how they can use this proposal.

A.4. Binary Transparency

So-called "Binary Transparency" may eventually allow users to verify that a program they’ve been delivered is one that’s available to the public, and not a specially-built version intended to attack just them. Binary transparency systems don’t exist yet, but they’re likely to work similarly to the successful Certificate Transparency logs described by [RFC6962].

Certificate Transparency depends on Signed Certificate Timestamps that prove a log contained a particular certificate at a particular time. To build the same thing for Binary Transparency logs containing HTTP resources or full websites, we’ll need a way to provide signatures of those resources, which signed exchanges provides.

A.5. Static Analysis

Native app stores like the Apple App Store [13] and the Android Play Store [14] grant their contents powerful abilities, which they attempt to make safe by analyzing the applications before offering them to people. The web has no equivalent way for people to wait to run an update of a web application until a trusted authority has vouched for it.

While full application analysis probably needs to wait until the authority can sign bundles of exchanges, authorities may be able to
guarantee certain properties by just checking a top-level resource and its [SRI]-constrained sub-resources.

A.6. Offline websites

Fully-offline websites can be represented as bundles of signed exchanges, although an optimization to reduce the number of signature verifications may be needed. Work on this is in progress in the https://github.com/WICG/webpackage [15] repository.

Appendix B. Requirements

B.1. Proof of origin

To verify that a thing came from a particular origin, for use in the same context as a TLS connection, we need someone to vouch for the signing key with as much verification as the signing keys used in TLS. The obvious way to do this is to re-use the web PKI and CA ecosystem.

B.1.1. Certificate constraints

If we re-use existing TLS server certificates, we incur the risks that:

1. TLS server certificates must be accessible from online servers, so they’re easier to steal or use as signing oracles than an offline key. An exchange’s signing key doesn’t need to be online.

2. A server using an origin-trusted key for one purpose (e.g. TLS) might accidentally sign something that looks like an exchange, or vice versa.

If these risks are too high, we could define a new Extended Key Usage (Section 4.2.1.12 of [RFC5280]) that requires CAs to issue new keys for this purpose or a new certificate extension to do the same. A new EKU would probably require CAs to also issue new intermediate certificates because of how browsers trust EKUs. Both an EKU and a new extension take a long time to deploy and allow CAs to charge exchange-signers more than normal server operators, which will reduce adoption.

The rest of this document assumes we can re-use existing TLS server certificates.
B.1.2. Signature constraints

In order to prevent an attacker who can convince the server to sign some resource from causing those signed bytes to be interpreted as something else, signatures here need to:

1. Avoid key types that are used for non-TLS protocols whose output could be confused with a signature. That may be just the "rsaEncryption" OID from [RFC8017].

2. Use the same format as TLS’s signatures, specified in Section 4.4.3 of [I-D.ietf-tls-tls13], with a context string that’s specific to this use.

The specification also needs to define which signing algorithm to use. It currently specifies that as a function from the key type, instead of allowing attacker-controlled data to specify it.

B.1.3. Retrieving the certificate

The client needs to be able to find the certificate vouching for the signing key, a chain from that certificate to a trusted root, and possibly other trust information like SCTs ([RFC6962]). One approach would be to include the certificate and its chain in the signature metadata itself, but this wastes bytes when the same certificate is used for multiple HTTP responses. If we decide to put the signature in an HTTP header, certificates are also unusually large for that context.

Another option is to pass a URL that the client can fetch to retrieve the certificate and chain. To avoid extra round trips in fetching that URL, it could be bundled (Appendix A.6) with the signed content or PUSHed (Appendix A.1) with it. The risks from the "client_certificate_url" extension (Section 11.3 of [RFC6066]) don’t seem to apply here, since an attacker who can get a client to load an exchange and fetch the certificates it references, can also get the client to perform those fetches by loading other HTML.

To avoid using an unintended certificate with the same public key as the intended one, the content of the leaf certificate or the chain should be included in the signed data, like TLS does (Section 4.4.3 of [I-D.ietf-tls-tls13]).

B.2. How much to sign

The previous [I-D.thomson-http-content-signature] and [I-D.burke-content-signature] schemes signed just the content, while ([I-D.cavage-http-signatures] could also sign the response headers
and the request method and path. However, the same path, response headers, and content may mean something very different when retrieved from a different server. Section 3.3 currently includes the whole request URL in the signature, but it’s possible we need a more flexible scheme to allow some higher-level protocols to accept a less-signed URL.

The question of whether to include other request headers—primarily the "accept" family—is still open. These headers need to be represented so that clients wanting a different language, say, can avoid using the wrong-language response, but it’s not obvious that there’s a security vulnerability if an attacker can spoof them. For now, the proposal (Section 3) omits other request headers.

In order to allow multiple clients to consume the same signed exchange, the exchange shouldn’t include the exact request headers that any particular client sends. For example, a Japanese resource wouldn’t include

accept-language: ja-JP, ja;q=0.9, en;q=0.8, zh;q=0.7, *;q=0.5

Instead, it would probably include just

accept-language: ja-JP, ja

and clients would use the same matching logic as for PUSH_PROMISE [16] frame headers.

B.2.1. Conveying the signed headers

HTTP headers are traditionally munged by proxies, making it impossible to guarantee that the client will see the same sequence of bytes as the author wrote. In the HTTPS world, we have more end-to-end header integrity, but it’s still likely that there are enough TLS-terminating proxies that the author’s signatures would tend to break before getting to the client.

There’s also no way in current HTTP for the response to a client-initiated request (Section 8.1 of [RFC7540]) to convey the request headers it expected to respond to. A PUSH_PROMISE (Section 8.2 of [RFC7540]) does not have this problem, and it would be possible to introduce a response header to convey the expected request headers.

Since proxies are unlikely to modify unknown content types, we can wrap the original exchange into an "application/http-exchange+cbor" format (Section 5) and include the "Cache-Control: no-transform" header when sending it.
To reduce the likelihood of accidental modification by proxies, the "application/http-exchange+cbor" format includes a file signature that doesn’t collide with other known signatures.

To help the PUSHed subresources use case (Appendix A.1), we might also want to extend the "PUSH_PROMISE" frame type to include a signature, and that could tell intermediates not to change the ensuing headers.

**B.3. Response lifespan**

A normal HTTPS response is authoritative only for one client, for as long as its cache headers say it should live. A signed exchange can be re-used for many clients, and if it was generated while a server was compromised, it can continue compromising clients even if their requests happen after the server recovers. This signing scheme needs to mitigate that risk.

**B.3.1. Certificate revocation**

Certificates are mis-issued and private keys are stolen, and in response clients need to be able to stop trusting these certificates as promptly as possible. Online revocation checks don’t work [17], so the industry has moved to pushed revocation lists and stapled OCSP responses [RFC6066].

Pushed revocation lists work as-is to block trust in the certificate signing an exchange, but the signatures need an explicit strategy to staple OCSP responses. One option is to extend the certificate download (Appendix B.1.3) to include the OCSP response too, perhaps in the TLS 1.3 CertificateEntry [18] format.

**B.3.2. Response downgrade attacks**

The signed content in a response might be vulnerable to attacks, such as XSS, or might simply be discovered to be incorrect after publication. Once the author fixes those vulnerabilities or mistakes, clients should stop trusting the old signed content in a reasonable amount of time. Similar to certificate revocation, I expect the best option to be stapled "this version is still valid" assertions with short expiration times.

These assertions could be structured as:

1. A signed minimum version number or timestamp for a set of request headers: This requires that signed responses need to include a version number or timestamp, but allows a server to provide a single signature covering all valid versions.
2. A replacement for the whole exchange’s signature. This requires the author to separately re-sign each valid version and requires each version to include a different update URL, but allows intermediates to serve less data. This is the approach taken in Section 3.

3. A replacement for the exchange’s signature and an update for the embedded "expires" and related cache-control HTTP headers [RFC7234]. This naturally extends authors’ intuitions about cache expiration and the existing cache revalidation behavior to signed exchanges. This is sketched and its downsides explored in Appendix C.

The signature also needs to include instructions to intermediates for how to fetch updated validity assertions.

Appendix C. Determining validity using cache control

This draft could expire signature validity using the normal HTTP cache control headers ([RFC7234]) instead of embedding an expiration date in the signature itself. This section specifies how that would work, and describes why I haven’t chosen that option.

The signatures in the "Signature" header field (Section 3.2) would no longer contain "date" or "expires" fields.

The validity-checking algorithm (Section 3.6) would initialize "date" from the resource’s "Date" header field (Section 7.1.1.2 of [RFC7231]) and initialize "expires" from either the "Expires" header field (Section 5.3 of [RFC7234]) or the "Cache-Control" header field’s "max-age" directive (Section 5.2.2.8 of [RFC7234]) (added to "date"), whichever is present, preferring "max-age" (or failing) if both are present.

Validity updates (Section 3.7) would include a list of replacement response header fields. For each header field name in this list, the client would remove matching header fields from the stored exchange’s response header fields. Then the client would append the replacement header fields to the stored exchange’s response header fields.

C.1. Example of updating cache control

For example, given a stored exchange of:
GET https://example.com/ HTTP/1.1
Accept: */*

HTTP/1.1 200
Date: Mon, 20 Nov 2017 10:00:00 UTC
Content-Type: text/html
Date: Tue, 21 Nov 2017 10:00:00 UTC
Expires: Sun, 26 Nov 2017 10:00:00 UTC

<!doctype html>
<html>
 ...
</html>

And an update listing the following headers:

Expires: Fri, 1 Dec 2017 10:00:00 UTC
Date: Sat, 25 Nov 2017 10:00:00 UTC

The resulting stored exchange would be:

GET https://example.com/ HTTP/1.1
Accept: */*

HTTP/1.1 200
Content-Type: text/html
Expires: Fri, 1 Dec 2017 10:00:00 UTC
Date: Sat, 25 Nov 2017 10:00:00 UTC

<!doctype html>
<html>
 ...
</html>

C.2. Downsides of updating cache control

In an exchange with multiple signatures, using cache control to expire signatures forces all signatures to initially live for the same period. Worse, the update from one signature’s "validityUrl" might not match the update for another signature. Clients would need to maintain a current set of headers for each signature, and then decide which set to use when actually parsing the resource itself.

This need to store and reconcile multiple sets of headers for a single signed exchange argues for embedding a signature’s lifetime into the signature.
Appendix D. Change Log

RFC EDITOR PLEASE DELETE THIS SECTION.

draft-02

- Signatures identify a header (e.g. Digest or MI) to guard the payload’s integrity instead of directly signing over the payload.
- The validityUrl is signed.
- Use CBOR maps where appropriate, and define how they’re canonicalized.
- Remove the update.url field from signature validity updates, in favor of just re-fetching the original request URL.
- Define an HTTP/2 extension to use a setting to enable cross-origin Server Push.
- Define an "Accept-Signature" header to negotiate whether to send Signatures and which ones.
- Define an "application/http-exchange+cbor" format to fetch signed exchanges without HTTP/2 Push.
- 2 new use cases.

Appendix E. Acknowledgements

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

Author’s Address

Jeffrey Yasskin
Google

Email: jyasskin@chromium.org