Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: July 27, 2019

J. Yasskin
Google

January 23, 2019

Signed HTTP Exchanges
draft-yasskin-http-origin-signed-responses-05

Abstract

This document specifies how a server can send an HTTP exchange--a request URL, content negotiation information, and a response--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 27, 2019.

Table of Contents

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:

- 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.

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

- 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

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

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

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

Exchange (noun) An HTTP request URL, content negotiation information, and an HTTP response. This can be encoded into a request message from a client with its matching response from a server, into the request in a PUSH_PROMISE with its matching response stream, or into the dedicated format in Section 5.3, which uses [I-D.ietf-httpbis-variants] to encode the content negotiation information. This is not quite the same meaning as defined by Section 8 of [RFC7540], which assumes the content negotiation information is embedded into HTTP request headers.

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

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

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

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. Signing an exchange

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

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

Each signature is parameterised with information to let a client fetch assurance that a signed exchange is still valid, in the face of revoked certificates and newly-discovered vulnerabilities. This

assurance can be bundled back into the signed exchange and forwarded to another client, which won't have to re-fetch this validity information for some period of time.

3.1. The Signature Header

The "Signature" header field conveys a 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 exchange's URL and response headers and identifies one of those headers that enforces the integrity of the exchange's payload.

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

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

Each parameterised identifier in the list MUST have parameters named "sig", "integrity", "validity-url", "date", and "expires". Each parameterised identifier MUST also have either "cert-url" and "cert-sha256" parameters or an "ed25519key" parameter. This specification gives no meaning to the identifier itself, which can be used as a human-readable identifier for the signature (see Section 3.1.2, Paragraph 1). The present parameters MUST have the following values:

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

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

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

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

"ed25519key" Byte sequence (Section 3.10 of [I-D.ietf-httpbis-header-structure]) holding an Ed25519 public key ([RFC8032]).

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

"date" and "expires" An integer (Section 3.6 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=*MEUCIQDXlI2gN3RNBlgFiuRNFpZXcDIaUpX6HIEwcZEc0cZYLAIga9DsVOMM+g5YpwEBdGW3sS+bvnmAJJiSMwhuBdc
  integrity="digest/mi-sha256";
  validity-url="https://example.com/resource.validity.1511128380";
  cert-url="https://example.com/oldcerts";
  cert-sha256=*W7uB969dFW3Mb5ZefPS9Tq5ZbH5iSmOILpjv2qEArmI=*;
  date=1511128380; expires=1511733180,
 sig2;
  sig=*MEQCIGjZRqTRf9iKNkGFyzRMTFgwf/BrY2ZNIP/dykhUV0aYAiBTXg+8wujoT4n/
W+cNgb7pGqQvIUGYZ8u8HZJ5YH26Qg=*;
  integrity="digest/mi-sha256";
  validity-url="https://example.com/resource.validity.1511128380";
  cert-url="https://example.com/newcerts";
  cert-sha256=*J/lEm9kNRODdCmINbvitpvdYKNQ+YgBj99DlYp4fEXw=*;
  date=1511128380; expires=1511733180,
 srisig;

sig=*lGZVaJJM5f2oGczFlLmBdKTDL+QADza4BgeO494ggACYJOvrof6uh5OJCcwKrk7DK+LBch0jssDYPp5CLc1SDA=*
  integrity="digest/mi-sha256";
  validity-url="https://example.com/resource.validity.1511128380";
  ed25519key=*zsSevyFsxyZHiUluVBDd4eypdRLTqyWRVOJuuKUz+A8=*
  date=1511128380; expires=1511733180,
 thirdpartysig;

sig=*MEYCIQCNxJzn6Rh2fNxsobktir8TkiaJYQFhWTuWI1i4PewQaQIhAMs2TVjc4rTshDtXbgQEOwgj2mRXALhfXPztXgF
  integrity="digest/mi-sha256";
  validity-url="https://thirdparty.example.com/resource.validity.1511161860";
```

```
  cert-url="https://thirdparty.example.com/certs";
  cert-sha256=*UeOwUPkvxlGRTyvHcsMUN0A2oNsZbU8EUvg8A9ZAnNc=*;
  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 "Digest" response header with the mi-sha256 digest algorithm to guard the integrity of the response payload.

The signatures include 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 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 publisher might be using two certificates because their readers have disjoint sets of roots in their trust stores.

The publisher 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 publisher 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.

3.1.2. Open Questions

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

Should the cert-url and validity-url 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.

3.2. CBOR representation of exchange response headers

To sign an exchange's response headers, they need to be serialized into a byte string. Since intermediaries and distributors (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 3.4), and then signed.

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

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

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

3.2.1. Example

Given the HTTP exchange:

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

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

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

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

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

3.3. Loading a certificate chain

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

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

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

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

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

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

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

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

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

3. Return "certificate-chain".

3.4. Canonical CBOR serialization

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

- Integers and the lengths of arrays, maps, and strings MUST use the smallest possible encoding.
- Items MUST NOT be encoded with indefinite length.
- The keys in every map MUST be sorted in the bytewise lexicographic order of their canonical encodings. For example, the following keys are correctly sorted:
  1. 10, encoded as 0A.
  2. 100, encoded as 18 64.
  3. -1, encoded as 20.
  4. "z", encoded as 61 7A.
  5. "aa", encoded as 62 61 61.
  6. [100], encoded as 81 18 64.
  7. [-1], encoded as 81 20.
  8. false, encoded as F4.

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

3.5. Signature validity

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

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

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

Potentially-valid results include:

- The signed headers of the exchange so that higher-level protocols can avoid relying on unsigned headers, and
- Either a certificate chain or a public key so that a higher-level protocol can determine whether it's actually valid.

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

1. Let:
    * "signature" be the signature (byte sequence in the parameterised identifier's "sig" parameter).
    * "integrity" be the signature's "integrity" parameter.
    * "validity-url" be the signature's "validity-url" parameter.
    * "cert-url" be the signature's "cert-url" parameter, if any.
    * "cert-sha256" be the signature's "cert-sha256" parameter, if any.
    * "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.

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

    1. If "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 [RFC8446].

        6. Otherwise, either return "invalid" or set "signing-alg" to a non-legacy signing algorithm defined by TLS 1.3 or later ([RFC8446]). This choice MUST depend only on "publicKey"'s type and not on any other context.

    2. If "ed25519key" is present, set "publicKey" to "ed25519key" and "signing-alg" to ed25519, as defined by [RFC8032]

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

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

5. Let "message" be the concatenation of the following byte strings. This matches the [RFC8446] 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".

Note: RFC EDITOR PLEASE DELETE THIS NOTE; The implementation of the final RFC MUST use this context string, but implementations of drafts MUST NOT use it and MUST use another draft-specific string beginning with "HTTP Exchange 1 " instead. This ensures that signers can predict how their signatures will be used.

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

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

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

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

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

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

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

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

   Note that this intentionally differs from TLS 1.3, which signs the entire certificate chain in its Certificate Verify (Section 4.4.3 of [RFC8446]), in order to allow updating the stapled OCSP response without updating signatures at the same time.

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

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

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

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

9. If "integrity" names a header field and parameter that is not present in "responseHeaders" 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". If the selected header field provides integrity guarantees weaker than SHA-256, return "invalid". If validating integrity using the selected header field requires the client to process records larger than 16384 bytes, return "invalid". Clients MUST implement at least the "Digest" header field with its "mi-sha256" digest algorithm (Section 3 of [I-D.thomson-http-mice]).

   Note: RFC EDITOR PLEASE DELETE THIS NOTE; Implementations of drafts of this RFC MUST recognize the draft spelling of the content encoding and digest algorithm specified by [I-D.thomson-http-mice] until that draft is published as an RFC. For example, implementations of draft-thomson-http-mice-03 would use "mi-sha256-03" and MUST NOT use "mi-sha256" itself. This ensures that final implementations don't need to handle compatibility with implementations of early drafts of that content encoding.

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

10. Return "potentially-valid" with whichever is present of "certificate-chain" or "ed25519key".

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

3.5.1. Open Questions

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

3.6. Updating signature validity

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

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

The "validity-url" parameter (Paragraph 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 "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 4.2.6 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).

```
Signature:
 sig1;
  sig=*MEUCIQ...*;
  ...
  validity-url="https://example.com/resource.validity.1511157180";
  cert-url="https://example.com/oldcerts";
  date=1511128380; expires=1511733180,
 sig2;
  sig=*MEQCIG...*;
  ...
  validity-url="https://example.com/resource.validity.1511157180";
  cert-url="https://example.com/newcerts";
  date=1511128380; expires=1511733180,
 thirdpartysig;
  sig=*MEYCIQ...*;
  ...
  validity-url="https://thirdparty.example.com/resource.validity.1511161860";
  cert-url="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 exipire until 23:11 that night, so the client needs to fetch "https://example.com/resource.validity.1511157180" (the "validity-url" of "sig1" and "sig2") if it wishes to update those signatures. This URL might contain:

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

This indicates that the client could fetch a newer version at "https://example.com/resource" (the original URL of the exchange), or that the validity period of the old version can be extended by replacing the first two of the original signatures (the ones with a validity-url 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...*;
  ...
  validity-url="https://example.com/resource.validity.1511157180";
  cert-url="https://example.com/newcerts";
  date=1511733180; expires=1512337980,
 thirdpartysig;
  sig=*MEYCIQ...*;
  ...
  validity-url="https://thirdparty.example.com/resource.validity.1511161860";
  cert-url="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.

3.7. 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 3.4 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 "digest/" indicate that the client supports the "Digest" header field ({{!RFC3230) with the parameter from the HTTP Digest Algorithm Values Registry [6] registry named in lower-case by the rest of the identifier. For example, "digest/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/mi-sha256" 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 lower-case 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 a byte sequence (Section 3.10 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 or end with the "*"s that indicate a byte sequence.

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.

3.7.4. Examples

Accept-Signature: digest/mi-sha256

states that the client will accept signatures with payload integrity assured by the "Digest" header and "mi-sha256" digest algorithm and implies that the client will accept 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 "Digest: mi-sha256" header field.

3.7.5. 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 identifiers indicating feature support?

4. Cross-origin trust

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

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

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

- "payload", a stream of bytes constituting the exchange's payload body (Section 3.3 of [RFC7230]).

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

1. If the signature's "validity-url" parameter (Paragraph 6) is not same-origin [7] with "requestUrl", return "invalid".

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

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

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

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

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

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

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

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

    3. Validate that "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].

8. Return "valid".

4.1. Uncached header fields

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

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

4.1.1. Stateful header fields

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

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

- "Authentication-Control", [RFC8053]
- "Authentication-Info", [RFC7615]
- "Clear-Site-Data", [W3C.WD-clear-site-data-20171130]
- "Optional-WWW-Authenticate", [RFC8053]

- "Proxy-Authenticate", [RFC7235]
- "Proxy-Authentication-Info", [RFC7615]
- "Public-Key-Pins", [RFC7469]
- "Sec-WebSocket-Accept", [RFC6455]
- "Set-Cookie", [RFC6265]
- "Set-Cookie2", [RFC2965]
- "SetProfile", [W3C.NOTE-OPS-OverHTTP]
- "Strict-Transport-Security", [RFC6797]
- "WWW-Authenticate", [RFC7235]

4.2. Certificate Requirements

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

```
id-ce-canSignHttpExchanges OBJECT IDENTIFIER ::= { TBD }

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 [RFC8446]).

RFC EDITOR PLEASE DELETE THE REST OF THE PARAGRAPHS IN THIS SECTION

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

Implementations of drafts of this specification MAY recognize the "id-ce-canSignHttpExchangesDraft" OID as identifying the CanSignHttpExchanges extension. 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, clients don't know how the server's content negotiation algorithm works, and intermediate servers add response header fields, it can be impossible to have a signature for the exchange's exact request, content negotiation, and response. Therefore, when a client calls the validation procedure in Section 3.5) to validate the "Signature" header field for an exchange represented as a normal HTTP request/response pair, it MUST pass:

- The "Signature" header field,
- The effective request URI (Section 5.5 of [RFC7230]) of the request,
- The serialized headers defined by Section 5.1.1, and
- The response's payload.

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.

If the signed response includes a "Variants" header field, the client MUST use the cache behavior algorithm in Section 4 of [I-D.ietf-httpbis-variants] to check that the signed response is an appropriate representation for the request the client is trying to

fulfil. If the response is not an appropriate representation, the client MUST treat the signature as invalid.

5.1.1. Serialized headers for a same-origin response

The serialized 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 canonical serialization (Section 3.4) of the CBOR representation (Section 3.2) of the response status code (Section 6 of [RFC7231]) and the response header fields whose names are listed in that response's "Signed-Headers" header field (Section 5.1.2). If a response header field name from "Signed-Headers" does not appear in the response's header fields, the exchange has no serialized 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 serialized headers.

5.1.1.1. Open Questions

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

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 3.8 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

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.

5.2.1. 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.

5.2.2. 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).

5.2.2.1. Open Questions

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

5.2.3. 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], unless there is another error as described below.

Instead, the client MUST validate such a PUSH_PROMISE and its response against the following list:

1. If the PUSH_PROMISE includes any non-pseudo request header fields, the client MUST treat it as a stream error (Section 5.4.2 of [RFC7540]) of type PROTOCOL_ERROR.

2. If the PUSH_PROMISE's method is not "GET", the client MUST treat it as a stream error (Section 5.4.2 of [RFC7540]) of type PROTOCOL_ERROR.

3. Run the algorithm in Section 4 over:

   * The "Signature" header field from the response.

   * The effective request URI from the PUSH_PROMISE.

   * The canonical serialization (Section 3.4) of the CBOR representation (Section 3.2) of the pushed response's status and its headers except for the "Signature" header field.

   * The response's payload.

   If this returns "invalid", 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.

5.2.3.1. Open Questions

Is it right that "validity-url" 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.

5.3. application/signed-exchange format

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

When served over HTTP, a response containing an "application/signed-exchange" payload MUST include at least the following response header fields, to reduce content sniffing vulnerabilities (Section 6.8):

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

- X-Content-Type-Options: nosniff

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

1. 8 bytes consisting of the ASCII characters "sxg1" followed by 4 0x00 bytes, 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: RFC EDITOR PLEASE DELETE THIS NOTE; The implementation of the final RFC MUST use this file signature, but implementations of drafts MUST NOT use it and MUST use another implementation-specific 8-byte string beginning with "sxg1-".

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

3. "fallbackUrlLength" bytes holding a "fallbackUrl", which MUST be an absolute URL with a scheme of "https".

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

   Issue: Should this fallback information also include the method?

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

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

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

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

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

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

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

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

5.3.2. Example

An example "application/signed-exchange" file representing a possible signed exchange with https://example.com/ [8] 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\0\0\0\0<2-byte length of the following url string>
https://example.com/<3-byte length of the following header
value><3-byte length of the encoding of the
following map>sig1; sig=*...; integrity="digest/mi-sha256"; ...{
  ':status': '200',
  'content-type': 'text/html'
}<!doctype html>\r\n<html>...
```

5.3.3. Open Questions

Should this be a CBOR format, or is the current mix of binary and CBOR better?

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

6. Security considerations

6.1. Over-signing

If a publisher blindly signs all responses as their origin, they can cause at least two kinds of problems, described below. To avoid this, publishers SHOULD design their systems to opt particular public content that doesn't depend on authentication status into signatures instead of signing by default.

Signing systems SHOULD also incorporate the following mitigations to reduce the risk that private responses are signed:

1. Strip the "Cookie" request header field and other identifying information like client authentication and TLS session IDs from requests whose exchange is destined to be signed, before forwarding the request to a backend.

2. Only sign exchanges where the response includes a "Cache-Control: public" header. Clients are not required to fail signature-checking for exchanges that omit this "Cache-Control" response header field to reduce the risk that naive signing systems blindly add it.

6.1.1. Session fixation

Blind signing can sign responses that create session cookies or otherwise change state on the client to identify a particular session. This breaks certain kinds of CSRF defense and can allow an attacker to force a user into the attacker's account, where the user might unintentionally save private information, like credit card numbers or addresses.

This specification defends against cookie-based attacks by blocking the "Set-Cookie" response header, but it cannot prevent Javascript or other response content from changing state.

6.1.2. Misleading content

If a site signs private information, an attacker might set up their own account to show particular private information, forward that signed information to a victim, and use that victim's confusion in a more sophisticated attack.

Stripping authentication information from requests before sending them to backends is likely to prevent the backend from showing attacker-specific information in the signed response. It does not prevent the attacker from showing their victim a signed-out page when

the victim is actually signed in, but while this is still misleading, it seems less likely to be useful to the attacker.

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 [RFC8336] 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. Publishers should consider shorter signature expiration times than they use for cache expiration times.

Clients MAY also check the "validity-url" (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 5.2.3.1 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 MUST 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 a publisher 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"? This is mitigated by using

[I-D.ietf-httpbis-variants] instead of request headers to describe how the client should run content negotiation.

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 3.5 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 5.2 requires all headers to be included in the signature before trusting cross-origin pushed resources, at Ryan Sleevi's recommendation.

6.6. application/signed-exchange

Clients MUST NOT trust an effective request URI claimed by an "application/signed-exchange" resource (Section 5.3) without either ensuring the resource was transferred from a server that was authoritative (Section 9.1 of [RFC7230]) for that URI's origin, or calling the algorithm in Section 5.3.1 and getting "valid" back.

6.7. Key re-use with TLS

In general, key re-use across multiple protocols is a bad idea.

Using an exchange-signing key in a TLS (or other directly-internet-facing) server increases the risk that an attacker can steal the private key, which will allow them to mint packages (similar to Section 6.4) until their theft is discovered.

Using a TLS key in a CanSignHttpExchanges certificate makes it less likely that the server operator will discover key theft, due to the considerations in Section 6.2.

This specification uses the CanSignHttpExchanges X.509 extension (Section 4.2) to discourage re-use of TLS keys to sign exchanges or vice-versa.

We require that clients reject certificates with the CanSignHttpExchanges extension when making TLS connections to minimize the chance that servers will re-use keys like this. Ideally, we would make the extension critical so that even clients that don't understand it would reject such TLS connections, but this proved impossible because certificate-validating libraries ship on significantly different schedules from the clients that use them.

Even once all clients reject these certificates in TLS connections, this will still just discourage and not prevent key re-use, since a server operator can unwisely request two different certificates with the same private key.

6.8. Content sniffing

While modern browsers tend to trust the "Content-Type" header sent with a resource, especially when accompanied by "X-Content-Type-Options: nosniff", plugins will sometimes search for executable content buried inside a resource and execute it in the context of the origin that served the resource, leading to XSS vulnerabilities. For example, some PDF reader plugins look for "%PDF" anywhere in the first 1kB and execute the code that follows it.

The "application/signed-exchange" format (Section 5.3) includes a URL and response headers early in the format, which an attacker could use to cause these plugins to sniff a bad content type.

To avoid vulnerabilities, in addition to the response header requirements in Section 5.3, servers are advised to only serve an "application/signed-exchange" resource (SXG) from a domain if it would also be safe for that domain to serve the SXG's content directly, and to follow at least one of the following strategies:

1. Only serve signed exchanges from dedicated domains that don't have access to sensitive cookies or user storage.

2. Generate signed exchanges "offline", that is, in response to a trusted author submitting content or existing signatures reaching a certain age, rather than in response to untrusted-reader queries.

3. Do all of:

    1. If the SXG's fallback URL (Section 5.3) is derived from the request URL, percent-encode [9] ([URL]) any bytes that are greater than 0x7E or are not URL code points [10] ([URL]) in the fallback URL . It is particularly important to make sure no unescaped nulls (0x00) or angle brackets (0x3C and 0x3E) appear.

    2. Do not reflect request header fields into the set of response headers.

There are still a few binary length fields that an attacker may influence to contain sensitive bytes, but they're always followed by lowercase alphabetic strings from a small set of possibilities, which

reduces the chance that a client will sniff them as indicating a particular content type.

To encourage servers to include the "X-Content-Type-Options: nosniff" header field, clients SHOULD reject signed exchanges served without it.

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 ([RFC8446]) or a successor transport layer, and MUST NOT accept exchanges transferred over plain HTTP without TLS.

8. IANA considerations

TODO: possibly register the validity-url 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 3.1 of this document

8.2. Accept-Signature Header Field Registration

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

Header field name: "Accept-Signature"

Applicable protocol: http

Status: standard

Author/Change controller: IETF

Specification document(s): Section 3.7 of this document

8.3. Signed-Headers Header Field Registration

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

Header field name: "Signed-Headers"

Applicable protocol: http

Status: standard

Author/Change controller: IETF

Specification document(s): Section 5.1.2 of this document

8.4. 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

8.5. 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

8.6. Internet Media Type application/signed-exchange

Type name: application

Subtype name: signed-exchange

Required parameters:

o v: A string denoting the version of the file format. ([RFC5234] ABNF: "version = DIGIT/%x61-7A") The version defined in this specification is "1". 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: RFC EDITOR PLEASE DELETE THIS NOTE; Implementations of drafts of this specification MUST NOT use simple integers to describe their versions, and MUST instead define implementation-specific strings to identify which draft is implemented. The newest version of [I-D.yasskin-httpbis-origin-signed-exchanges-impl] describes the meaning of one such string.

Optional parameters: N/A

Encoding considerations: binary

Security considerations: see Section 6.6

Interoperability considerations: N/A

Published specification: This specification (see Section 5.3).

Applications that use this media type: N/A

Fragment identifier considerations: N/A

Additional information:

Deprecated alias names for this type: N/A

Magic number(s): 73 78 67 31 00

File extension(s): .sxg

Macintosh file type code(s): N/A

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

Intended usage: COMMON

Restrictions on usage: N/A

Author: See Authors' Addresses section.

Change controller: IESG

8.7. Internet Media Type application/cert-chain+cbor

Type name: application

Subtype name: cert-chain+cbor

Required parameters: N/A

Optional parameters: N/A

Encoding considerations: binary

Security considerations: N/A

Interoperability considerations: N/A

Published specification: This specification (see Section 3.3).

Applications that use this media type: N/A

Fragment identifier considerations: N/A

Additional information:

Deprecated alias names for this type: N/A

Magic number(s): 1*9(??) 67 F0 9F 93 9C E2 9B 93

File extension(s): N/A

Macintosh file type code(s): N/A

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

Intended usage: COMMON

Restrictions on usage: N/A

Author: See Authors' Addresses section.

Change controller: IESG

9. References

9.1. Normative References

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

[I-D.ietf-cbor-cddl] Birkholz, H., Vigano, C., and C. Bormann, "Concise data definition language (CDDL): a notational convention to express CBOR and JSON data structures", draft-ietf-cbor-cddl-06 (work in progress), November 2018.

[I-D.ietf-httpbis-header-structure] Nottingham, M. and P. Kamp, "Structured Headers for HTTP", draft-ietf-httpbis-header-structure-09 (work in progress), December 2018.

[I-D.ietf-httpbis-variants] Nottingham, M., "HTTP Representation Variants", draft-ietf-httpbis-variants-04 (work in progress), October 2018.

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

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

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC3230] Mogul, J. and A. Van Hoff, "Instance Digests in HTTP", RFC 3230, DOI 10.17487/RFC3230, January 2002, <https://www.rfc-editor.org/info/rfc3230>.

[RFC3864] Klyne, G., Nottingham, M., and J. Mogul, "Registration Procedures for Message Header Fields", BCP 90, RFC 3864, DOI 10.17487/RFC3864, September 2004, <https://www.rfc-editor.org/info/rfc3864>.

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

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

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

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

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

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

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

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

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

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

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

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

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

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] Cavage, M. and M. Sporny, "Signing HTTP Messages", draft-cavage-http-signatures-10 (work in progress), May 2018.

[I-D.ietf-httpbis-cache]
Fielding, R., Nottingham, M., and J. Reschke, "HTTP Caching", draft-ietf-httpbis-cache-03 (work in progress), October 2018.

[I-D.ietf-httpbis-http2-secondary-certs]
Bishop, M., Sullivan, N., and M. Thomson, "Secondary Certificate Authentication in HTTP/2", draft-ietf-httpbis-http2-secondary-certs-03 (work in progress), October 2018.

[I-D.thomson-http-content-signature]
Thomson, M., "Content-Signature Header Field for HTTP", draft-thomson-http-content-signature-00 (work in progress), July 2015.

[I-D.yasskin-httpbis-origin-signed-exchanges-impl]
Yasskin, J. and K. Ueno, "Signed HTTP Exchanges Implementation Checkpoints", draft-yasskin-httpbis-origin-signed-exchanges-impl-02 (work in progress), September 2018.

[I-D.yasskin-webpackage-use-cases]
Yasskin, J., "Use Cases and Requirements for Web Packages", draft-yasskin-webpackage-use-cases-01 (work in progress), March 2018.

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

[RFC6066] Eastlake 3rd, D., "Transport Layer Security (TLS) Extensions: Extension Definitions", RFC 6066, DOI 10.17487/RFC6066, January 2011, <https://www.rfc-editor.org/info/rfc6066>.

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

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

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

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

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

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

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

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

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

[RFC8336] Nottingham, M. and E. Nygren, "The ORIGIN HTTP/2 Frame", RFC 8336, DOI 10.17487/RFC8336, March 2018, <https://www.rfc-editor.org/info/rfc8336>.

[SRI] Akhawe, D., Braun, F., Marier, F., and J. Weinberger, "Subresource Integrity", World Wide Web Consortium Recommendation REC-SRI-20160623, June 2016, <http://www.w3.org/TR/2016/REC-SRI-20160623>.

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

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

9.3. URIs

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

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

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

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

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

[6] https://www.iana.org/assignments/http-dig-alg/http-dig-alg.xhtml

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

[8] https://example.com/

[9] https://url.spec.whatwg.org/#percent-encode

[10] https://url.spec.whatwg.org/#url-code-points

[11] https://calendar.perfplanet.com/2013/big-bad-preloader/

[12] https://github.com/mikewest/signature-based-sri

[13] https://github.com/mikewest/signature-based-sri/issues/5

[14] https://www.apple.com/ios/app-store/

[15] https://play.google.com/store

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

[17] https://www.imperialviolet.org/2012/02/05/crlsets.html

[18] https://tlswg.github.io/tls13-spec/draft-ietf-tls-tls13.html#ocsp-and-sct

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 logicalsrc="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 [11] 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 [12] 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 [13]. 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 [14] and the Android Play Store [15] 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 [16] 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.

These risks are considered too high, so we define a new X.509 certificate extension in Section 4.2 that requires CAs to issue new certificates for this purpose. We expect at least one low-cost CA to be willing to sign certificates with this extension.

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 the new X.509 extension here is forbidden from being used in TLS servers. If Section 4.2 changes to allow re-use in TLS servers, we would 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 [RFC8446], 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 [RFC8446]).

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 5.1.1 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.

Servers might want to sign other request headers in order to capture their effects on content negotiation. However, there's no standard algorithm to check that a client's actual request headers match request headers sent by a server. The most promising attempt at this is [I-D.ietf-httpbis-variants], which encodes the content negotiation algorithm into the "Variants" and "Variant-Key" response headers. The proposal here (Section 3) assumes that is in use and doesn't sign request 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 publisher published. 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 publisher's signatures would tend to break before getting to the client.

There's 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, but we sidestep that by conveying content negotiation information in response headers, per [I-D.ietf-httpbis-variants].

Since proxies are unlikely to modify unknown content types, we can wrap the original exchange into an "application/signed-exchange" format (Section 5.3) and include the "Cache-Control: no-transform" header when sending it.

To reduce the likelihood of accidental modification by proxies, the "application/signed-exchange" 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 publisher 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 publishers' 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.

B.4. Low implementation complexity

Simpler implementations are, all things equal, less likely to include bugs. This section describes decisions that were made in the rest of the specification to reduce complexity.

B.4.1. Limited choices

In general, we're trying to eliminate unnecessary choices in the specification. For example, instead of requiring clients to support two methods for verifying payload integrity, we only require one.

B.4.2. Bounded-buffering integrity checking

Clients can be designed with a more-trusted network layer that decides how to trust resources and then provides those resources to less-trusted rendering processes along with handles to the storage and other resources they're allowed to access. If the network layer can enforce that it only operates on chunks of data up to a certain size, it can avoid the complexity of spooling large files to disk.

To allow the network layer to verify signed exchanges using a bounded amount of memory, Section 5.3 requires the signature to be less than 16kB and the headers to be less than 512kB, and Section 3.5 requires that the MI record size be less than 16kB. This allows the network layer to validate a bounded chunk at a time, and pass that chunk on to a renderer, and then forget about that chunk before processing the next one.

The "Digest" header field from [RFC3230] requires the network layer to buffer the entire response body, so it's disallowed.

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 3.1) would no longer contain "date" or "expires" fields.

The validity-checking algorithm (Section 3.5) 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 3.6) 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  / HTTP/1.1
Host: example.com
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>
...
```

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 / HTTP/1.1
Host: example.com
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>
...
```

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 "validity-url" 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-05

- Define absolute URLs, and limit the schemes each instance can use.
- Fill in TBD size limits.
- Update to mice-03 including the Digest header.
- Refer to draft-yasskin-httpbis-origin-signed-exchanges-impl for draft version numbers.
- Require "exchange"'s response to be cachable by a shared cache.
- Define the "integrity" field of the Signature header to include subfields of the main integrity-protecting header, including the digest algorithm.
- Put a fallback URL at the beginning of the "application/signed-exchange" format, which replaces the ':url' key from the CBOR representation of the exchange's request and response metadata and headers.
- Remove the rest of the request headers from the signed data, in favor of representing content negotiation with the "Variants" response header.
- Make the signed message format a concatenation of byte sequences, which helps implementations avoid re-serializing the exchange's request and response metadata and headers.

- Explicitly check the response payload's integrity instead of assuming the client did it elsewhere in processing the response.
- Reject uncached header fields.
- Update to draft-ietf-httpbis-header-structure-09.
- Update to the final TLS 1.3 RFC.

draft-04

- Update to draft-ietf-httpbis-header-structure-06.
- Replace the application/http-exchange+cbor format with a simpler application/signed-exchange format that:
  - Doesn't require a streaming CBOR parser parse it from a network stream.
  - Doesn't allow request payloads or response trailers, which don't fit into the signature model.
  - Allows checking the signature before parsing the exchange headers.
- Require absolute URLs.
- Make all identifiers in headers lower-case, as required by Structured Headers.
- Switch back to the TLS 1.3 signature format.
- Include the version and draft number in the signature context string.
- Remove support for integrity protection using the Digest header field.
- Limit the record size in the mi-sha256 encoding.
- Forbid RSA keys, and only require clients to support secp256r1 keys.
- Add a test OID for the CanSignHttpExchanges X.509 extension.

draft-03

- Allow each method of transferring an exchange to define which headers are signed, have the cross-origin methods use all headers, and remove the "allResponseHeaders" flag.

- Describe footguns around signing private content, and block certain headers to make it less likely.

- Define a CBOR structure to hold the certificate chain instead of re-using the TLS1.3 message. The TLS 1.3 parser fails on unexpected extensions while this format should ignore them, and apparently TLS implementations don't expose their message parsers enough to allow passing a message to a certificate verifier.

- Require an X.509 extension for the signing certificate.

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 Devin Mullins, 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