HTTP
Internet-Draft
Intended status: Standards Track
Expires: September 10, 2020

R. Polli
Team Digitale, Italian Government
L. Pardue
Cloudflare

March 09, 2020

Digest Headers
draft-ietf-httpbis-digest-headers-02

Abstract

This document defines the Digest and Want-Digest header fields for HTTP, thus allowing client and server to negotiate an integrity checksum of the exchanged resource representation data.

This document obsoletes RFC 3230. It replaces the term "instance" with "representation", which makes it consistent with the HTTP Semantic and Context defined in RFC 7231.

Note to Readers

RFC EDITOR: please remove this section before publication

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 at https://github.com/httpwg/http-extensions [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 September 10, 2020.

Table of Contents

1. Introduction

The core specification of HTTP does not define a means to protect the integrity of resources. When HTTP messages are transferred between endpoints, the protocol might choose to make use of features of the lower layer in order to provide some integrity protection; for instance TCP checksums or TLS records [RFC2818].

However, there are cases where relying on this alone is insufficient. An HTTP-level integrity mechanism that operates independent of transfer can be used to detect programming errors and/or corruption of data at rest, be used across multiple hops in order to provide end-to-end integrity guarantees, aid fault diagnosis across hops and

system boundaries, and can be used to validate integrity when reconstructing a resource fetched using different HTTP connections.

This document defines a mechanism that acts on HTTP representation-data. It can be combined with other mechanisms that protect representation-metadata, such as digital signatures, in order to protect the desired parts of an HTTP exchange in whole or in part.

1.1. A Brief History of Integrity Header Fields

The Content-MD5 header field was originally introduced to provide integrity, but HTTP/1.1 ([RFC7231], Appendix B) obsoleted it:

> The Content-MD5 header field has been removed because it was inconsistently implemented with respect to partial responses.

[RFC3230] provided a more flexible solution introducing the concept of "instance", and the header fields "Digest" and "Want-Digest".

1.2. This Proposal

The concept of "selected representation" defined in [RFC7231] made [RFC3230] definitions inconsistent with the current standard. A refresh was then required.

This document updates the "Digest" and "Want-Digest" header field definitions to align with [RFC7231] concepts.

This approach can be easily adapted to use-cases where the transferred data does require some sort of manipulation to be considered a representation or conveys a partial representation of a resource (eg. Range Requests [RFC7233]).

Changes are semantically compatible with existing implementations and better cover both the request and response cases.

The value of "Digest" is calculated on selected representation, which is tied to the value contained in any "Content-Encoding" or "Content-Type" header fields. Therefore, a given resource may have multiple different digest values.

To allow both parties to exchange a Digest of a representation with no content codings [3] two more algorithms are added ("ID-SHA-256" and "ID-SHA-512").

1.3. Goals

The goals of this proposal are:

1. Digest coverage for either the resource's "representation data" or "selected representation data" communicated via HTTP.

2. Support for multiple digest algorithms.

3. Negotiation of the use of digests.

The goals do not include:

HTTP Message integrity: The digest mechanism described here does not cover the full HTTP message nor its semantic, as representation metadata are not included in the checksum.

Header field integrity: The digest mechanisms described here cover only representation and selected representation data, and do not protect the integrity of associated representation metadata or other message header fields.

Authentication: The digest mechanisms described here are not meant to support authentication of the source of a digest or of a message or anything else. These mechanisms, therefore, are not a sufficient defense against many kinds of malicious attacks.

Privacy: Digest mechanisms do not provide message privacy.

Authorization: The digest mechanisms described here are not meant to support authorization or other kinds of access controls.

1.4. Notational Conventions

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] and [RFC8174]) when, and only when, they appear in all capitals, as shown here.

This document uses the Augmented BNF defined in [RFC5234] and updated by [RFC7405] along with the "#rule" extension defined in Section 7 of [RFC7230].

The definitions "representation", "selected representation", "representation data", "representation metadata", and "payload body" in this document are to be interpreted as described in [RFC7230] and [RFC7231].

The definition "validator" in this document is to be interpreted as described in Section 7.2 of [RFC7231].

2. Representation Digest

The representation digest is an integrity mechanism for HTTP resources which uses a checksum that is calculated independently of the payload body and message body. It uses the representation data (see [RFC7231]), that can be fully or partially contained in the message body, or not contained at all:

```
representation-data := Content-Encoding( Content-Type( bits ) )
```

This takes into account the effect of the HTTP semantics on the messages; for example the payload body can be affected by Range Requests or methods such as HEAD, while the message body is dependent on transfer codings and other transformations: Appendix A contains several examples to help illustrate those effects.

A representation digest consists of the value of a checksum computed on the entire selected "representation data" of a resource together with an indication of the algorithm used (and any parameters)

```
representation-data-digest = digest-algorithm "="
                             <encoded digest output>
```

The checksum is computed using one of the "digest-algorithms" listed in Section 5 and then encoded in the associated format.

The example below shows the "sha-256" digest-algorithm which uses base64 encoding.

```
sha-256=X48E9qOokqqrvdts8nOJRJN3OWDUoyWxBf7kbu9DBPE=
```

3. The Digest Header Field

The Digest header field contains a list of one or more representation digest values as defined in Section 2. It can be used in both request and response.

```
Digest = "Digest" ":" OWS 1#representation-data-digest
```

The resource is specified by the effective request URI and any "validator" contained in the message.

The relationship between Content-Location (see [RFC7231] Section 3.1.4.2) and Digest is demonstrated in Section 9.7. A comprehensive set of examples showing the impacts of representation

metadata, payload transformations and HTTP methods on digest is provided in Section 9 and Section 10.

A Digest header field MAY contain multiple representation-data-digest values. This could be useful for responses expected to reside in caches shared by users with different browsers, for example.

A recipient MAY ignore any or all of the representation-data-digests in a Digest header field. This allows the recipient to choose which digest-algorithm(s) to use for validation instead of verifying every received representation-data-digest.

A sender MAY send a representation-data-digest using a digest-algorithm without knowing whether the recipient supports the digest-algorithm, or even knowing that the recipient will ignore it.

Two examples of its use are

```
Digest: id-sha-512=WZDPaVn/7XgHaAy8pmojAkGWoRx2UFChF41A2svX+TaPm
                   AbwAgBWnrIiYllu7BNNyealdVLvRwE\nmTHWXvJwew==

Digest: sha-256=4REjxQ4yrqUVicfSKYNO/cF9zNj5ANbzgDZt3/h3Qxo=,
        id-sha-256=X48E9qOokqqrvdts8nOJRJN3OWDUoyWxBf7kbu9DBPE=
```

4. The Want-Digest Header Field

The Want-Digest message header field indicates the sender's desire to receive a representation digest on messages associated with the request URI and representation metadata.

```
Want-Digest = "Want-Digest" ":" OWS 1#want-digest-value
want-digest-value = digest-algorithm [ ";" "q" "=" qvalue]
qvalue = ( "0"  [ "."  0*1DIGIT ] ) /
         ( "1"  [ "."  0*1( "0" ) ] )
```

If a digest-algorithm is not accompanied by a qvalue, it is treated as if its associated qvalue were 1.0.

The sender is willing to accept a digest-algorithm if and only if it is listed in a Want-Digest header field of a message, and its qvalue is non-zero.

If multiple acceptable digest-algorithm values are given, the sender's preferred digest-algorithm is the one (or ones) with the highest qvalue.

Two examples of its use are

```
Want-Digest: sha-256
Want-Digest: SHA-512;q=0.3, sha-256;q=1, md5;q=0
```

5. Digest Algorithm Values

Digest algorithm values are used to indicate a specific digest computation. For some algorithms, one or more parameters can be supplied.

```
digest-algorithm = token
```

The BNF for "parameter" is as is used in [RFC7230]. All digest-algorithm values are case-insensitive.

The Internet Assigned Numbers Authority (IANA) acts as a registry for digest-algorithm values. The registry contains the tokens listed below.

Some algorithms, although registered, have since been found vulnerable: the MD5 algorithm MUST NOT be used due to collision attacks [CMU-836068] and the SHA algorithm is NOT RECOMMENDED due to collision attacks [IACR-2019-459].

SHA-256

* Description: The SHA-256 algorithm [RFC6234]. The output of this algorithm is encoded using the base64 encoding [RFC4648].
* Reference: [RFC6234], [RFC4648], this document.
* Status: standard

SHA-512

* Description: The SHA-512 algorithm [RFC6234]. The output of this algorithm is encoded using the base64 encoding [RFC4648].
* Reference: [RFC6234], [RFC4648], this document.
* Status: standard

MD5

* Description: The MD5 algorithm, as specified in [RFC1321]. The output of this algorithm is encoded using the base64 encoding

[RFC4648]. The MD5 algorithm MUST NOT be used as it's now vulnerable to collision attacks [CMU-836068].

* Reference: [RFC1321], [RFC4648], this document.

* Status: deprecated

SHA

* Description: The SHA-1 algorithm [RFC3174]. The output of this algorithm is encoded using the base64 encoding [RFC4648]. The SHA algorithm is NOT RECOMMENDED as it's now vulnerable to collision attacks [IACR-2019-459].

* Reference: [RFC3174], [RFC6234], [RFC4648], this document.

* Status: obsoleted

UNIXsum

* Description: The algorithm computed by the UNIX "sum" command, as defined by the Single UNIX Specification, Version 2 [UNIX]. The output of this algorithm is an ASCII decimal-digit string representing the 16-bit checksum, which is the first word of the output of the UNIX "sum" command.

* Reference: [UNIX], this document.

* Status: standard

UNIXcksum

* Description: The algorithm computed by the UNIX "cksum" command, as defined by the Single UNIX Specification, Version 2 [UNIX]. The output of this algorithm is an ASCII digit string representing the 32-bit CRC, which is the first word of the output of the UNIX "cksum" command.

* Reference: [UNIX], this document.

* Status: standard

To allow sender and recipient to provide a checksum which is independent from "Content-Encoding", the following additional algorithms are defined:

ID-SHA-512

* Description: The sha-512 digest of the representation-data of the resource when no content coding is applied (eg. "Content-Encoding: identity")

* Reference: [RFC6234], [RFC4648], this document.

* Status: standard

ID-SHA-256

* Description: The sha-256 digest of the representation-data of the resource when no content coding is applied (eg. "Content-Encoding: identity")

* Reference: [RFC6234], [RFC4648], this document.

* Status: standard

If other digest-algorithm values are defined, the associated encoding MUST either be represented as a quoted string, or MUST NOT include ";" or "," in the character sets used for the encoding.

6. Use of Digest when acting on resources

POST and PATCH requests can appear to convey partial representations but are semantically acting on resources. The enclosed representation, including its metadata refers to that action.

In these requests the representation digest MUST be computed on the representation-data of that action. This is the only possible choice because representation digest requires complete representation metadata (see Section 2).

In responses,

o if the representation describes the status of the request, "Digest" MUST be computed on the enclosed representation (see Section 9.8);

o if there is a referenced resource "Digest" MUST be computed on the selected representation of the referenced resource even if that is different from the target resource. That might or might not result in computing "Digest" on the enclosed representation.

The latter case might be done according to the HTTP semantics of the given method, for example using the "Content-Location" header field. In contrast, the "Location" header field does not affect "Digest" because it is not representation metadata.

6.1. Digest and PATCH

In PATCH requests the representation digest MUST be computed on the patch document because the representation metadata refers to the patch document and not to the target resource (see Section 2 of [RFC5789]).

In PATCH responses the representation digest MUST be computed on the selected representation of the patched resource.

"Digest" usage with PATCH is thus very similar to the POST one, but with the resource's own semantic partly implied by the method and by the patch document.

7. Deprecate Negotiation of Content-MD5

This RFC deprecates the negotiation of Content-MD5 as it has been obsoleted by [RFC7231].

8. Relationship to Subresource Integrity (SRI)

Subresource Integrity [SRI] is an integrity mechanism that shares some similarities to the present document's mechanism. However, there are differences in motivating factors, threat model and specification of integrity digest generation, signalling and validation.

SRI allows a first-party authority to declare an integrity assertion on a resource served by a first or third party authority. This is done via the "integrity" attribute that can be added to "script" or "link" HTML elements. Therefore, the integrity assertion is always made out-of-band to the resource fetch. In contrast, the "Digest" header field is supplied in-band alongside the selected representation, meaning that an authority can only declare an integrity assertion for itself. Methods to improve the security properties of representation digests are presented in Section 11. This contrast is interesting because on one hand self-assertion is less likely to be affected by coordination problems such as the first-party holding stale information about the third party, but on the other hand the self-assertion is only as trustworthy as the authority that provided it.

The SRI "integrity" attribute contains a cryptographic hash algorithm and digest value which is similar to "representation-data-digest" (see Section 2). The major differences are in serialization format.

The SRI digest value is calculated over the identity encoding of the resource, not the selected representation (as specified for "representation-data-digest" in this document). Section 3.4.5 of [SRI] describes the benefit of the identity approach - the SRI "integrity" attribute can contain multiple algorithm-value pairs where each applies to a different identity encoded payload. This allows for protection of distinct resources sharing a URL. However, this is a contrast to the design of representation digests, where multiple "Digest" field-values all protect the same representation.

SRI does not specify handling of partial representation data (e.g. Range requests). In contrast, this document specifies handling in terms that are fully compatible with core HTTP concepts (an example is provided in Section 9.3).

SRI specifies strong requirements on the selection of algorithm for generation and validation of digests. In contrast, the requirements in this document are weaker.

SRI defines no method for a client to declare an integrity assertion on resources it transfers to a server. In contrast, the "Digest" header field can appear on requests.

8.1. Supporting Both SRI and Representation Digest

The SRI and Representation Digest mechanisms are different and complementary but one is not capable of replacing the other because they have different threat, security and implementation properties.

A user agent that supports both mechanisms is expected to apply the rules specified for each but since the two mechanisms are independent, the ordering is not important. However, a user agent supporting both could benefit from performing representation digest validation first because it does not always require a conversion into identity encoding.

There is a chance that a user agent supporting both mechanisms may find one validates successfully while the other fails. This document specifies no requirements or guidance for user agents that experience such cases.

9. Examples of Unsolicited Digest

The following examples demonstrate interactions where a server responds with a "Digest" header field even though the client did not solicit one using "Want-Digest".

9.1. Server Returns Full Representation Data

Request:

```
GET /items/123
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Encoding: identity
Digest: sha-256=X48E9qOokqqrvdts8nOJRJN3OWDUoyWxBf7kbu9DBPE=

{"hello": "world"}
```

9.2. Server Returns No Representation Data

As there is no content coding applied, the "sha-256" and the "id-sha-256" digest-values are the same.

Request:

```
HEAD /items/123
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Encoding: identity
Digest: id-sha-256=X48E9qOokqqrvdts8nOJRJN3OWDUoyWxBf7kbu9DBPE=
```

9.3. Server Returns Partial Representation Data

Request:

```
GET /items/123
Range: bytes=1-7
```

Response:

```
HTTP/1.1 206 Partial Content
Content-Type: application/json
Content-Encoding: identity
Content-Range: bytes 1-7/18
Digest: sha-256=X48E9qOokqqrvdts8nOJRJN3OWDUoyWxBf7kbu9DBPE=

"hello"
```

9.4. Client and Server Provide Full Representation Data

The request contains a "Digest" header calculated on the enclosed representation.

It also includes an "Accept-Encoding: br" header field that advertises the client supports brotli encoding.

The response includes a "Content-Encoding: br" that indicates the selected representation is brotli encoded. The "Digest" field-value is therefore different compared to the request.

The response body is displayed as a base64-encoded string because it contains non-printable characters.

Request:

```
PUT /items/123
Content-Type: application/json
Content-Encoding: identity
Accept-Encoding: br
Digest: sha-256=X48E9qOokqqrvdts8nOJRJN3OWDUoyWxBf7kbu9DBPE=

{"hello": "world"}
```

Response:

```
Content-Type: application/json
Content-Encoding: br
Digest: sha-256=4REjxQ4yrqUVicfSKYNO/cF9zNj5ANbzgDZt3/h3Qxo=

iwiAeyJoZWxsbyI6ICJ3b3JsZCJ9Aw==
```

9.5. Client Provides Full Representation Data, Server Provides No Representation Data

Request "Digest" value is calculated on the enclosed payload. Response "Digest" value depends on the representation metadata header fields, including "Content-Encoding: br" even when the response does not contain a payload body.

Request:

```
PUT /items/123
Content-Type: application/json
Content-Encoding: identity
Content-Length: 18
Accept-Encoding: br
Digest: sha-256=X48E9qOokqqrvdts8nOJRJN3OWDUoyWxBf7kbu9DBPE=

{"hello": "world"}
```

Response:

```
HTTP/1.1 204 No Content
Content-Type: application/json
Content-Encoding: br
Digest: sha-256=4REjxQ4yrqUVicfSKYNO/cF9zNj5ANbzgDZt3/h3Qxo=
```

9.6. Client and Server Provide Full Representation Data, Client Uses id-sha-256.

The response contains two digest values:

- one with no content coding applied, which in this case accidentally matches the unencoded digest-value sent in the request;
- one taking into account the "Content-Encoding".

As the response body contains non-printable characters, it is displayed as a base64-encoded string.

Request:

```
PUT /items/123 HTTP/1.1
Content-Type: application/json
Content-Encoding: identity
Accept-Encoding: br
Digest: sha-256=X48E9qOokqqrvdts8nOJRJN3OWDUoyWxBf7kbu9DBPE=

{"hello": "world"}

Response:

HTTP/1.1 200 OK
Content-Type: application/json
Content-Encoding: br
Digest: sha-256=4REjxQ4yrqUVicfSKYNO/cF9zNj5ANbzgDZt3/h3Qxo=,
        id-sha-256=X48E9qOokqqrvdts8nOJRJN3OWDUoyWxBf7kbu9DBPE=

iwiAeyJoZWxsbyI6ICJ3b3JsZCJ9Aw==
```

9.7. POST Response does not Reference the Request URI

Request "Digest" value is computed on the enclosed representation (see Section 6).

The representation enclosed in the response refers to the resource identified by "Content-Location" (see [RFC7231] Section 3.1.4.2 and Section 3.1.4.1 point 4).

"Digest" is thus computed on the enclosed representation.

Request:

```
POST /books HTTP/1.1
Content-Type: application/json
Accept: application/json
Accept-Encoding: identity
Digest: sha-256=bWopGGNiZtbVgHsG+I4knzfEJpmmmQHf7RHDXA3o1hQ=

{"title": "New Title"}
```

Response

```
HTTP/1.1 201 Created
Content-Type: application/json
Digest: id-sha-256=BZlF2v0IzjuxN01RQ97EUXriaNNLhtI8Chx8Eq+XYSc=
Content-Location: /books/123

{"id": "123", "title": "New Title"}
```

Note that a "204 No Content" response without a payload body but with the same "Digest" field-value would have been legitimate too.

9.8. POST Response Describes the Request Status

Request "Digest" value is computed on the enclosed representation (see Section 6).

The representation enclosed in the response describes the status of the request, so "Digest" is computed on that enclosed representation.

Response "Digest" has no explicit relation with the resource referenced by "Location".

Request:

```
POST /books HTTP/1.1
Content-Type: application/json
Accept: application/json
Accept-Encoding: identity
Digest: sha-256=bWopGGNiZtbVgHsG+I4knzfEJpmmmQHf7RHDXA3o1hQ=
Location: /books/123

{"title": "New Title"}
```

Response

```
HTTP/1.1 201 Created
Content-Type: application/json
Digest: id-sha-256=0o/WKwSfnmIoSlop2LV/ISaBDth05IeW27zzNMUh5l8=
Location: /books/123

{
  "status": "created",
  "id": "123",
  "ts": 1569327729,
  "instance": "/books/123"
}
```

9.9. Digest with PATCH

This case is analogous to a POST request where the target resource reflects the effective request URI.

The PATCH request uses the "application/merge-patch+json" media type defined in [RFC7396].

"Digest" is calculated on the enclosed payload, which corresponds to the patch document.

The response "Digest" is computed on the complete representation of the patched resource.

Request:

```
PATCH /books/123 HTTP/1.1
Content-Type: application/merge-patch+json
Accept: application/json
Accept-Encoding: identity
Digest: sha-256=bWopGGNiZtbVgHsG+I4knzfEJpmmmQHf7RHDXA3o1hQ=

{"title": "New Title"}
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json
Digest: id-sha-256=BZlF2v0IzjuxN01RQ97EUXriaNNLhtI8Chx8Eq+XYSc=

{"id": "123", "title": "New Title"}
```

Note that a "204 No Content" response without a payload body but with the same "Digest" field-value would have been legitimate too.

9.10. Error responses

In error responses, the representation-data does not necessarily refer to the target resource. Instead it refers to the representation of the error.

In the following example a client attempts to patch the resource located at /books/123. However, the resource does not exist and the server generates a 404 response with a body that describes the error in accordance with [RFC7807].

The digest of the response is computed on this enclosed representation.

Request:

```
PATCH /books/123 HTTP/1.1
Content-Type: application/merge-patch+json
Accept: application/json
Accept-Encoding: identity
Digest: sha-256=bWopGGNiZtbVgHsG+I4knzfEJpmmmQHf7RHDXA3o1hQ=

{"title": "New Title"}

Response:

HTTP/1.1 404 Not Found
Content-Type: application/problem+json
Digest: sha-256=UJSojgEzqUe4UoHzmNl5d2xkmrW3BOdmvsvWu1uFeu0=

{
  "title": "Not Found",
  "detail": "Cannot PATCH a non-existent resource",
  "status": 404
}
```

10. Examples of Want-Digest Solicited Digest

The following examples demonstrate interactions where a client solicits a "Digest" using "Want-Digest".

10.1. Server Selects Client's Least Preferred Algorithm

The client requests a digest, preferring sha. The server is free to reply with sha-256 anyway.

```
Request:

GET /items/123 HTTP/1.1
Want-Digest: sha-256;q=0.3, sha;q=1


Response:

HTTP/1.1 200 OK
Content-Type: application/json
Content-Encoding: identity
Digest: sha-256=X48E9qOokqqrvdts8nOJRJN3OWDUoyWxBf7kbu9DBPE=

{"hello": "world"}
```

10.2. Server Selects Algorithm Unsupported by Client

The client requests a sha digest only. The server is currently free to reply with a Digest containing an unsupported algorithm.

Request:

```
GET /items/123
Want-Digest: sha;q=1
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Encoding: identity
Digest: id-sha-512=WZDPaVn/7XgHaAy8pmojAkGWoRx2UFChF41A2svX+TaPm
                  +AbwAgBWnrIiYllu7BNNyealdVLvRwE\nmTHWXvJwew==

{"hello": "world"}
```

10.3. Server Does Not Support Client Algorithm and Returns an Error

The client requests a sha Digest, the server advises for sha-256 and sha-512

Request:

```
GET /items/123
Want-Digest: sha;q=1
```

Response:

```
HTTP/1.1 400 Bad Request
Want-Digest: sha-256, sha-512
```

11. Security Considerations

11.1. Digest Does Not Protect the Full HTTP Message

This document specifies a data integrity mechanism that protects HTTP "representation data", but not HTTP "representation metadata" header fields, from certain kinds of accidental corruption.

"Digest" is not intended as general protection against malicious tampering with HTTP messages, this can be achieved by combining it

with other approaches such as transport-layer security or digital signatures.

11.2. Broken Cryptographic Algorithms

Cryptographic algorithms are intended to provide a proof of integrity suited towards cryptographic constructions such as signatures.

However, these rely on collision-resistance for their security proofs [CMU-836068]. The MD5 and SHA-1 algorithms are vulnerable to collisions attacks, so MD5 MUST NOT be used and SHA-1 is NOT RECOMMENDED for use with "Digest".

11.3. Other Deprecated Algorithms

The ADLER32 algorithm defined in [RFC1950] has been deprecated by [RFC3309] because under certain conditions it provides weak detection of errors and is now NOT RECOMMENDED for use with "Digest".

11.4. Digest for End-to-End Integrity

"Digest" alone does not provide end-to-end integrity of HTTP messages over multiple hops, as it just covers the "representation data" and not the "representation metadata".

Besides, it allows to protect "representation data" from buggy manipulation, buggy compression, etc.

Moreover identity digest algorithms (eg. ID-SHA-256 and ID-SHA-512) allow piecing together a resource from different sources (e.g. different servers that perhaps apply different content codings) enabling the user-agent to detect that the application-layer tasks completed properly, before handing off to say the HTML parser, video player etc.

Even a simple mechanism for end-to-end validation is thus valuable.

11.5. Digest and Content-Location in responses

When a state-changing method returns the "Content-Location" header field, the enclosed representation refers to the resource identified by its value and "Digest" is computed accordingly.

11.6. Usage in signatures

Digital signatures are widely used together with checksums to provide the certain identification of the origin of a message [NIST800-32].

Such signatures can protect one or more header fields and there are additional considerations when "Digest" is included in this set.

Since the "Digest" header field is a hash of a resource representation, it explicitly depends on the "representation metadata" (eg. the values of "Content-Type", "Content-Encoding" etc). A signature that protects "Digest" but not other "representation metadata" can expose the communication to tampering. For example, an actor could manipulate the "Content-Type" field-value and cause a digest validation failure at the recipient, preventing the application from accessing the representation. Such an attack consumes the resources of both endpoints. See also Section 11.5.

"Digest" SHOULD always be used over a connection which provides integrity at the transport layer that protects HTTP header fields.

A "Digest" header field using NOT RECOMMENDED digest-algorithms SHOULD NOT be used in signatures.

11.7. Message Truncation

...

11.8. Algorithm Agility

...

12. IANA Considerations

12.1. Establish the HTTP Digest Algorithm Values

This memo sets this spec to be the establishing document for the HTTP Digest Algorithm Values [4]

12.2. The "status" Field in the HTTP Digest Algorithm Values

This memo adds the field "Status" to the HTTP Digest Algorithm Values [5] registry. The allowed values for the "Status" fields are described below.

Status Specify "standard", "experimental", "historic", "obsoleted", or "deprecated" according to the type and status of the primary document in which the algorithm is defined.

12.3. Deprecate "MD5" Digest Algorithm

This memo updates the "MD5" digest algorithm in the HTTP Digest Algorithm Values [6] registry:

- Digest Algorithm: MD5
- Description: As specified in Section 5.
- Status: As specified in Section 5.

12.4. Update "CRC32C" Digest Algorithm

This memo updates the "CRC32c" digest algorithm in the HTTP Digest Algorithm Values [7] registry:

- Digest Algorithm: CRC32c
- Description: The CRC32c algorithm is a 32-bit cyclic redundancy check. It achieves a better hamming distance (for better error-detection performance) than many other 32-bit CRC functions. Other places it is used include iSCSI and SCTP. The 32-bit output is encoded in hexadecimal (using between 1 and 8 ASCII characters from 0-9, A-F, and a-f; leading 0's are allowed). For example, CRC32c=0a72a4df and crc32c=A72A4DF are both valid checksums for the 3-byte message "dog".
- Reference: [RFC4960] appendix B, this document.
- Status: standard.

12.5. Obsolete "SHA" Digest Algorithm

This memo updates the "SHA" digest algorithm in the HTTP Digest Algorithm Values [8] registry:

- Digest Algorithm: SHA
- Description: As specified in Section 5.
- Status: As specified in Section 5.

12.6. Obsolete "ADLER32" Digest Algorithm

This memo updates the "ADLER32" digest algorithm in the HTTP Digest Algorithm Values [9] registry:

- Digest Algorithm: ADLER32

- Description: The ADLER32 algorithm is a checksum specified in [RFC1950] "ZLIB Compressed Data Format". The 32-bit output is encoded in hexadecimal (using between 1 and 8 ASCII characters from 0-9, A-F, and a-f; leading 0's are allowed). For example, ADLER32=03da0195 and ADLER32=3DA0195 are both valid checksums for the 4-byte message "Wiki". This algorithm is obsoleted and SHOULD NOT be used.

- Status: obsoleted

12.7. The "ID-SHA-256" Digest Algorithm

This memo registers the "ID-SHA-256" digest algorithm in the HTTP Digest Algorithm Values [10] registry:

- Digest Algorithm: ID-SHA-256

- Description: As specified in Section 5.

- Status: As specified in Section 5.

12.8. The "ID-SHA-512" Digest Algorithm

This memo registers the "ID-SHA-512" digest algorithm in the HTTP Digest Algorithm Values [11] registry:

- Digest Algorithm: ID-SHA-512

- Description: As specified in Section 5.

- Status: As specified in Section 5.

12.9. Changes compared to RFC5843

The status of "MD5" has been updated to "deprecated", and its description states that this algorithm MUST NOT be used.

The status of "SHA" has been updated to "obsoleted", and its description states that this algorithm is NOT RECOMMENDED.

The status for "CRC32C" has been updated to "standard".

The "ID-SHA-256" and "ID-SHA-512" algorithms have been added to the registry.

12.10. Want-Digest Header Field Registration

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

Header field name: "Want-Digest"

Applicable protocol: http

Status: standard

Author/Change controller: IETF

Specification document(s): Section 4 of this document

12.11. Digest Header Field Registration

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

Header field name: "Digest"

Applicable protocol: http

Status: standard

Author/Change controller: IETF

Specification document(s): Section 3 of this document

13. References

13.1. Normative References

[CMU-836068]
Carnagie Mellon University, Software Engineering Institute, "MD5 Vulnerable to collision attacks", December 2008, <https://www.kb.cert.org/vuls/id/836068/>.

[IACR-2019-459]
Leurent, G. and T. Peyrin, "From Collisions to Chosen-Prefix Collisions Application to Full SHA-1", May 2019, <https://eprint.iacr.org/2019/459.pdf>.

[NIST800-32]
National Institute of Standards and Technology, U.S. Department of Commerce, "Introduction to Public Key Technology and the Federal PKI Infrastructure", February 2001, <https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-32.pdf>.

[RFC1321] Rivest, R., "The MD5 Message-Digest Algorithm", RFC 1321, DOI 10.17487/RFC1321, April 1992, <https://www.rfc-editor.org/info/rfc1321>.

[RFC1950] Deutsch, P. and J-L. Gailly, "ZLIB Compressed Data Format Specification version 3.3", RFC 1950, DOI 10.17487/RFC1950, May 1996, <https://www.rfc-editor.org/info/rfc1950>.

[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>.

[RFC3174] Eastlake 3rd, D. and P. Jones, "US Secure Hash Algorithm 1 (SHA1)", RFC 3174, DOI 10.17487/RFC3174, September 2001, <https://www.rfc-editor.org/info/rfc3174>.

[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>.

[RFC3309] Stone, J., Stewart, R., and D. Otis, "Stream Control Transmission Protocol (SCTP) Checksum Change", RFC 3309, DOI 10.17487/RFC3309, September 2002, <https://www.rfc-editor.org/info/rfc3309>.

[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>.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <https://www.rfc-editor.org/info/rfc4648>.

[RFC4960] Stewart, R., Ed., "Stream Control Transmission Protocol", RFC 4960, DOI 10.17487/RFC4960, September 2007, <https://www.rfc-editor.org/info/rfc4960>.

[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>.

[RFC5843] Bryan, A., "Additional Hash Algorithms for HTTP Instance Digests", RFC 5843, DOI 10.17487/RFC5843, April 2010, <https://www.rfc-editor.org/info/rfc5843>.

[RFC6234] Eastlake 3rd, D. and T. Hansen, "US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)", RFC 6234, DOI 10.17487/RFC6234, May 2011, <https://www.rfc-editor.org/info/rfc6234>.

[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>.

[RFC7233] Fielding, R., Ed., Lafon, Y., Ed., and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Range Requests", RFC 7233, DOI 10.17487/RFC7233, June 2014, <https://www.rfc-editor.org/info/rfc7233>.

[RFC7405] Kyzivat, P., "Case-Sensitive String Support in ABNF", RFC 7405, DOI 10.17487/RFC7405, December 2014, <https://www.rfc-editor.org/info/rfc7405>.

[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>.

[UNIX] The Open Group, "The Single UNIX Specification, Version 2 - 6 Vol Set for UNIX 98", February 1997.

13.2. Informative References

[RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, DOI 10.17487/RFC2818, May 2000, <https://www.rfc-editor.org/info/rfc2818>.

[RFC5789] Dusseault, L. and J. Snell, "PATCH Method for HTTP", RFC 5789, DOI 10.17487/RFC5789, March 2010, <https://www.rfc-editor.org/info/rfc5789>.

[RFC7396] Hoffman, P. and J. Snell, "JSON Merge Patch", RFC 7396, DOI 10.17487/RFC7396, October 2014, <https://www.rfc-editor.org/info/rfc7396>.

[RFC7807] Nottingham, M. and E. Wilde, "Problem Details for HTTP APIs", RFC 7807, DOI 10.17487/RFC7807, March 2016, <https://www.rfc-editor.org/info/rfc7807>.

[SRI] Akhawe, D., Braun, F., Marier, F., and J. Weinberger, "Subresource Integrity", W3C Recommendation REC-SRI-20160623, June 2016, <https://www.w3.org/TR/2016/REC-SRI-20160623/>.

13.3. URIs

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

[2] https://github.com/httpwg/http-extensions

[3] https://tools.ietf.org/html/rfc7231#section-3.1.2.1

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

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

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

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

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

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

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

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

Appendix A. Resource Representation and Representation-Data

The following examples show how representation metadata, payload transformations and method impacts on the message and payload body. When the payload body contains non-printable characters (eg. when it is compressed) it is shown as base64-encoded string.

Here is a gzip-compressed json object

Request:

```
PUT /entries/1234 HTTP/1.1
Content-Type: application/json
Content-Encoding: gzip

H4sIAItWyFwC/6tWSlSyUlAypANQqgUAREcqfG0AAAA=
```

Now the same payload body conveys a malformed json object.

Request:

```
PUT /entries/1234 HTTP/1.1
Content-Type: application/json

H4sIAItWyFwC/6tWSlSyUlAypANQqgUAREcqfG0AAAA=
```

A Range-Request alters the payload body, conveying a partial representation.

Request:

```
GET /entries/1234 HTTP/1.1
Range: bytes=1-7
```

Response:

```
HTTP/1.1 206 Partial Content
Content-Encoding: gzip
Content-Type: application/json
Content-Range: bytes 1-7/18

iwgAla3RXA==
```

Now the method too alters the payload body.

Request:

```
HEAD /entries/1234 HTTP/1.1
Accept: application/json
Accept-Encoding: gzip
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Encoding: gzip
```

Finally the semantics of an HTTP response might decouple the effective request URI from the enclosed representation. In the example response below, the "Content-Location" header field indicates that the enclosed representation refers to the resource available at "/authors/123".

Request:

```
POST /authors/ HTTP/1.1
Accept: application/json
Content-Type: application/json

{"author": "Camilleri"}
```

Response:

```
HTTP/1.1 201 Created
Content-Type: application/json
Content-Location: /authors/123
Location: /authors/123

{"id": "123", "author": "Camilleri"}
```

Appendix B. FAQ

1. Why remove all references to content-md5?

   Those were unnecessary to understanding and using this spec.

2. Why remove references to instance manipulation?

   Those were unnecessary for correctly using and applying the spec. An example with Range Request is more than enough. This doc uses the term "partial representation" which should group all those cases.

3. How to use "Digest" with "PATCH" method?

   See Section 6.

4. Why remove references to delta-encoding?

Unnecessary for a correct implementation of this spec. The revised spec can be nicely adapted to "delta encoding", but all the references here to delta encoding don't add anything to this RFC. Another job would be to refresh delta encoding.

5. Why remove references to Digest Authentication?

   This RFC seems to me completely unrelated to Digest Authentication but for the word "Digest".

6. What changes in "Want-Digest"?

   We allow to use the "Want-Digest" in responses to advertise the supported digest-algorithms and the inability to accept requests with unsupported digest-algorithms.

7. Does this spec changes supported algorithms?

   This RFC updates [RFC5843] which is still delegated for all algorithms updates, and adds two more algorithms: ID-SHA-256 and ID-SHA-512 which allows to send a checksum of a resource representation with no content codings applied.

Acknowledgements

The vast majority of this document is inherited from [RFC3230], so thanks to J. Mogul and A. Van Hoff for their great work. The original idea of refreshing this document arose from an interesting discussion with M. Nottingham, J. Yasskin and M. Thomson when reviewing the MICE content coding.

Code Samples

RFC Editor: Please remove this section before publication.

How can I generate and validate the Digest values shown in the examples throughout this document?

The following python3 code can be used to generate digests for json objects using SHA algorithms for a range of encodings. Note that these are formatted as base64. This function could be adapted to other algorithms and should take into account their specific formatting rules.

```
import base64, json, hashlib, brotli


def digest(item, encoding=lambda x: x, algorithm=hashlib.sha256):
    json_bytes = json.dumps(item).encode()
    content_encoded = encoding(json_bytes)
    checksum_bytes = algorithm(content_encoded).digest()
    return base64.encodebytes(checksum_bytes).strip()


item = {"hello": "world"}

print("Encoding | digest-algorithm | digest-value")
print("Identity | sha256 |", digest(item))
# Encoding | digest-algorithm | digest-value
# Identity | sha256 | 4REjxQ4yrqUVicfSKYNO/cF9zNj5ANbzgDZt3/h3Qxo=

print("Encoding | digest-algorithm | digest-value")
print("Brotli | sha256 |", digest(item, encoding=brotli.compress))
# Encoding | digest-algorithm | digest-value
# Brotli , sha256 4REjxQ4yrqUVicfSKYNO/cF9zNj5ANbzgDZt3/h3Qxo=


print("Encoding | digest-algorithm | digest-value")
print("Identity | sha512 |", digest(item, algorithm=hashlib.sha512))
# Encoding | digest-algorithm | digest-value
# Identity | sha512 | b'WZDPaVn/7XgHaAy8pmojAkGWoRx2UFChF41A2s
vX+TaPm+AbwAgBWnrIiYllu7BNNyealdVLvRwE\nmTHWXvJwew==\n'
```

Changes

RFC Editor: Please remove this section before publication.

E.1. Since draft-ietf-httpbis-digest-headers-00

- Align title with document name
- Add id-sha-* algorithm examples #880
- Reference [RFC6234] and [RFC3174] instead of FIPS-1
- Deprecate MD5
- Obsolete ADLER-32 but don't forbid it #828
- Update CRC32C value in IANA table #828
- Use when acting on resources (POST, PATCH) #853

- Added Relationship with SRI, draft Use Cases #868, #971
- Warn about the implications of "Content-Location"

E.2. Since draft-ietf-httpbis-digest-headers-01

- Digest of error responses is computed on the error representation-data #1004
- Effect of HTTP semantics on payload and message body moved to appendix #1122
- Editorial refactoring, moving headers sections up. #1109-#1112, #1116, #1117, #1122-#1124

Authors' Addresses

Roberto Polli
Team Digitale, Italian Government

Email: robipolli@gmail.com

Lucas Pardue
Cloudflare

Email: lucaspardue.24.7@gmail.com