Workgroup: HTTP
Internet-Draft:
draft-ietf-httpbis-digest-headers-06
Obsoletes: 3230 (if approved)
Published: 27 September 2021
Intended Status: Standards Track
Expires: 31 March 2022
Authors: R. Polli
Team Digitale, Italian Government
L. Pardue
Cloudflare

Digest Fields

Abstract

This document defines HTTP fields that support integrity checksums. The Digest field can be used for the integrity of HTTP representations. The Content-Digest field can be used for the integrity of HTTP message content. Want-Digest and Want-Content-Digest can be used to indicate a sender's desire to receive integrity fields respectively.

This document obsoletes RFC 3230.

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/.

The source code and issues list for this draft can be found at https://github.com/httpwg/http-extensions.

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 31 March 2022.

Copyright Notice

Table of Contents

1. Introduction

HTTP does not define a means to protect the integrity of representations. 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].

This document defines two digest integrity mechanisms for HTTP. First, representation data integrity, which acts on representation data (Section 3.2 of [SEMANTICS]). Second, content digest integrity, which acts on conveyed content (Section 6.4 of [SEMANTICS]). Both mechanisms operate independent of transport integrity, offering the

potential to detect programming errors and corruption of data in flight or at rest. They can be used across multiple hops in order to provide end-to-end integrity guarantees, which can aid fault diagnosis when resources are transferred across hops and system boundaries. Finally, they can be used to validate integrity when reconstructing a resource fetched using different HTTP connections.

This document obsoletes [RFC3230].

1.1. Document Structure

This document is structured as follows:

- Section 2 describes concepts related to representation digests,
- Section 3 defines the Digest request and response header and trailer field,
- Section 4 defines the Content-Digest request and response header and trailer field,
- Section 5 defines the Want-Digest and Want-Content-Digest request and response header and trailer field,
- Section 6 and Appendix D.1 describe algorithms and their relation to Digest,
- Section 7 details computing representation digests,
- Appendix D.2 obsoletes Digest field parameters, and
- Appendix B and Appendix C provide examples of using Digest and Want-Digest.

1.2. Concept Overview

This document defines the Digest request and response header and trailer field; see Section 3. At a high level, the value contains a checksum, computed over selected representation data (Section 3.2 of [SEMANTICS]), that the recipient can use to validate integrity. Basing Digest on the selected representation makes it straightforward to apply it to use-cases where the transferred data requires some sort of manipulation to be considered a representation or conveys a partial representation of a resource, such as Range Requests (see Section 14.2 of [SEMANTICS]).

To support use-cases where a simple checksum of the content bytes is required, this document introduces the Content-Digest request and response header and trailer field; see Section 4.

Digest and Content-Digest support algorithm agility. The Want-Digest and Want-Content-Digest fields allows endpoints to express interest in Digest and Content-Digest respectively, and preference of algorithms in either.

Digest field calculations are tied to the Content-Encoding and Content-Type header fields. Therefore, a given resource may have multiple different checksum values when transferred with HTTP. To allow both parties to exchange a simple checksum with no content codings (see Section 8.4.1 of [SEMANTICS]), two more digest-algorithms are added ("id-sha-256" and "id-sha-512").

Digest fields do not provide integrity for HTTP messages or fields. However, they can be combined with other mechanisms that protect metadata, such as digital signatures, in order to protect the phases of an HTTP exchange in whole or in part.

This specification does not define means for authentication, authorization or privacy.

1.3. Replacing RFC 3230

Historically, the Content-MD5 header field provided an HTTP integrity mechanism but HTTP/1.1 ([RFC7231], Appendix B) obsoleted it due to inconsistent handling of partial responses. [RFC3230] defined the concept of "instance" digests and a more flexible integrity scheme to help address issues with Content-MD5. It first introduced the Digest and Want-Digest fields. HTTP terminology has evolved since [RFC3230] was published. The concept of "instance" has been superseded by selected representation.

This document replaces [RFC3230]. The Digest and Want-Digest field definitions are updated to align with the terms and notational conventions in [SEMANTICS]. Changes are intended to be semantically compatible with existing implementations but note that negotiation of Content-MD5 is deprecated Appendix D.1 and has been replaced by Content-Digest negotiation via Want-Content-Digest. Digest field parameters are obsoleted Appendix D.2 and the algorithm table has been updated to reflect the current state of the art.

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] [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 5.6.1 of [SEMANTICS] and the "qvalue" rule defined in Section 12.4.2 of [SEMANTICS].

The definitions "representation", "selected representation", "representation data", "representation metadata", and "content" in this document are to be interpreted as described in [SEMANTICS].

Algorithm names respect the casing used in their definition document (e.g. SHA-1, CRC32c) whereas digest-algorithm tokens are quoted (e.g. "sha", "crc32c").

2. Representation Digest

The representation digest is an integrity mechanism for HTTP resources which uses a checksum that is calculated independently of the content (see Section 6.4 of [SEMANTICS]). It uses the representation data (see Section 8.1 of [SEMANTICS]), that can be fully or partially contained in the content, or not contained at all.

This takes into account the effect of the HTTP semantics on the messages; for example, the content can be affected by Range Requests or methods such as HEAD, while the way the content is transferred "on the wire" is dependent on other transformations (e.g. transfer codings for HTTP/1.1 - see Section 6.1 of [HTTP11]). To help illustrate how such things affect Digest, several examples are provided in Appendix A.

A representation digest consists of the value of a checksum computed on the entire selected representation data (see Section 8.1 of [SEMANTICS]) of a resource identified according to Section 6.4.2 of [SEMANTICS] together with an indication of the algorithm used:

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

When a message has no representation data it is still possible to assert that no representation data was sent computing the representation digest on an empty string (see Section 8.3).

The checksum is computed using one of the digest-algorithms listed in the HTTP Digest Algorithm Values Registry (see Section 6) and then encoded in the associated format.

3. The Digest Field

The Digest field contains a comma-separated list of one or more representation digest values as defined in Section 2. It can be used in both requests and responses.

```
Digest = 1#representation-data-digest
```

For example:

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

A Digest field MAY contain multiple representation-data-digest values. For example, a server may provide representation-data-digest values using different algorithms, allowing it to support a population of clients with different evolving capabilities; this is particularly useful in support of transitioning away from weaker algorithms should the need arise (see Section 8.6).

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

A recipient MAY ignore any or all of the representation-data-digests in a Digest 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.

Digest can be sent in a trailer section. In this case, Digest MAY be merged into the header section; see Section 6.5.1 of [SEMANTICS].

When an incremental digest-algorithm is used, the sender and the receiver can dynamically compute the digest value while streaming the content.

A non-comprehensive set of examples showing the impacts of representation metadata, payload transformations and HTTP methods on Digest is provided in Appendix B and Appendix C.

4. The Content-Digest Field

The Content-Digest field contains a comma-separated list of one or more content digest values. A content digest value is computed by applying a digest-algorithm to the actual message content (see Section 6.4 of [SEMANTICS]). It can be used in both requests and responses.

```
Content-Digest = 1#content-digest
content-digest = digest-algorithm "="
                 <encoded digest output>
```

For example:

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

A Content-Digest field MAY contain multiple content-digest values, similarly to Digest (see Section 3)

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

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

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

Content-Digest can be sent in a trailer section. In this case, Content-Digest MAY be merged into the header section; see Section 6.5.1 of [SEMANTICS].

When an incremental digest-algorithm is used, the sender and the receiver can dynamically compute the digest value while streaming the content.

5. Want-Digest and Want-Content-Digest Fields

Senders can indicate their integrity checksum preferences using the Want-Digest or Want-Content-Digest fields. These can be used in both requests and responses.

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

Want-Content-Digest indicates the sender's desire to receive a content digest on messages associated with the request URI and representation metadata, using the Content-Digest field.

```
Want-Digest = 1#want-digest-value
Want-Content-Digest = 1#want-digest-value
want-digest-value = digest-algorithm [ ";" "q" "=" qvalue]
```

qvalue indicates the sender's digest-algorithm preferences. Section 12.4.2 of [SEMANTICS]) describes qvalue usage and semantics.

Senders can provide multiple digest-algorithm items with the same qvalue.

Examples:

```
Want-Digest: sha-256
Want-Digest: sha-512;q=0.3, sha-256;q=1, unixsum;q=0
Want-Content-Digest: sha-256
Want-Content-Digest: sha-512;q=0.3, sha-256;q=1, unixsum;q=0
```

6. Digest Algorithm Values

Digest-algorithm values are used to indicate a specific digest computation.

```
digest-algorithm = token
```

All digest-algorithm token values are case-insensitive but lower case is preferred; digest-algorithm token values MUST be compared in a case-insensitive fashion.

Every digest-algorithm defines its computation procedure and encoding output. Unless specified otherwise, comparison of encoded output is case-sensitive.

The "HTTP Digest Algorithm Values Registry", maintained by IANA at https://www.iana.org/assignments/http-dig-alg/ registers digest-algorithm values. Registrations MUST include the following fields:

- *Digest algorithm: the token value. The registry can be used to reserve token values

- *Status: the status of the algorithm. Use "standard" for standardized algorithms without known problems; "experimental" or some other appropriate value

  - -e.g. according to the type and status of the primary document in which the algorithm is defined; "deprecated" when the algorithm is insecure or otherwise undesirable; "reserved" when Digest algorithm references a reserved token value

- *Description: the description of the digest-algorithm and its encoding

- *Reference: a set of pointers to the primary documents defining the digest-algorithm

The associated encoding for new digest-algorithms MUST either be represented as a quoted string or MUST NOT include ";" or "," in the character sets used for the encoding.

Deprecated digest algorithms MUST NOT be used.

The registry is initialized with the tokens listed below.

sha-512

*Digest Algorithm: 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

sha-256

*Digest Algorithm: 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

md5

*Digest Algorithm: md5

*Description: The MD5 algorithm, as specified in [RFC1321]. The output of this algorithm is encoded using the base64 encoding [RFC4648]. This digest-algorithm is now vulnerable to collision attacks. See [NO-MD5] and [CMU-836068].

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

*Status: deprecated

sha

*Digest Algorithm: sha

*Description: The SHA-1 algorithm [RFC3174]. The output of this algorithm is encoded using the base64 encoding [RFC4648]. This digest-algorithm is now vulnerable to collision attacks. See [NO-SHA1] and [IACR-2020-014].

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

*Status: deprecated

unixsum

*Digest Algorithm: 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: deprecated

unixcksum

*Digest Algorithm: 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: deprecated

adler32

*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.

*Reference: [RFC1950], this document.

*Status: deprecated

crc32c

*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: deprecated.

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

id-sha-512

*Description: The sha-512 digest of the representation data of the resource when no content coding is applied

*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

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

*Status: standard

7. Using Digest in State-Changing Requests

When the representation enclosed in a state-changing request does not describe the target resource, the representation digest MUST be computed on the representation-data. This is the only possible choice because representation digest requires complete representation metadata (see Section 2).

In responses,

- if the representation describes the status of the request, Digest MUST be computed on the enclosed representation (see Appendix B.8);
- 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 is done according to the HTTP semantics of the given method, for example using the Content-Location header field (see Section 8.7 of [SEMANTICS]). In contrast, the Location header field does not affect Digest because it is not representation metadata.

For example, in PATCH requests, the representation digest will 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 [PATCH]). In responses, instead, the representation digest will be computed on the selected representation of the patched resource.

7.1. 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. An example is given in Appendix B.7.

8. Security Considerations

8.1. Digest Does Not Protect the Full HTTP Message

This document specifies a data integrity mechanism that protects HTTP representation data or content, but not HTTP header and trailer fields, from certain kinds of accidental corruption.

Digest fields are not intended to be a 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.

8.2. Digest for End-to-End Integrity

Digest fields can help detect representation data or content modification due to implementation errors, undesired "transforming proxies" (see Section 7.7 of [SEMANTICS]) or other actions as the data passes across multiple hops or system boundaries. Even a simple mechanism for end-to-end representation data integrity is valuable because user-agent can validate that resource retrieval succeeded before handing off to a HTML parser, video player etc. for parsing.

Identity digest-algorithms (e.g. "id-sha-256" and "id-sha-512") are particularly useful for end-to-end integrity because they allow piecing together a resource from different sources with different HTTP messaging characteristics. For example, different servers that apply different content codings.

Note that using digest fields alone does not provide end-to-end integrity of HTTP messages over multiple hops, since metadata could be manipulated at any stage. Methods to protect metadata are discussed in Section 8.3.

8.3. 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 HTTP fields and there are additional considerations when Digest is included in this set.

Since digest fields are hashes of resource representations, they explicitly depend on the representation metadata (e.g. 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 7.1.

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

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

Using signatures to protect the checksum of an empty representation allows receiving endpoints to detect if an eventual payload has been stripped or added.

Any mangling of digest fields, including de-duplication of representation-data-digest values or combining different field values (see Section 5.2 of [SEMANTICS]) might affect signature validation.

8.4. Usage in Trailer Fields

Before sending digest fields in a trailer section, the sender should consider that intermediaries are explicitly allowed to drop any trailer (see Section 6.5.2 of [SEMANTICS]).

When digest fields are used in a trailer section, the field-values are received after the content. Eager processing of content before the trailer section prevents digest validation, possibly leading to processing of invalid data.

Not every digest-algorithm is suitable for use in the trailer section, some may require to pre-process the whole payload before sending a message (e.g. see [I-D.thomson-http-mice]).

8.5. Usage with Encryption

Digest fields may expose details of encrypted payload when the checksum is computed on the unencrypted data. For example, the use of the "id-sha-256" digest-algorithm in conjunction with the encrypted content-coding [RFC8188].

The checksum of an encrypted payload can change between different messages depending on the encryption algorithm used; in those cases its value could not be used to provide a proof of integrity "at rest" unless the whole (e.g. encoded) content is persisted.

8.6. Algorithm Agility

The security properties of digest-algorithms are not fixed. Algorithm Agility (see [RFC7696]) is achieved by providing implementations with flexibility choose digest-algorithms from the IANA Digest Algorithm Values registry in Section 9.1.

To help endpoints distinguish weaker algorithms from stronger ones, this document adds to the IANA Digest Algorithm Values registry a new "Status" field containing the most recent appraisal of the digest-algorithm.

An endpoint might have a preference for algorithms, such as preferring "standard" algorithms over "deprecated" ones. Transition

from weak algorithms is supported by negotiation of digest-algorithm using Want-Digest or Want-Content-Digest (see Section 5) or by sending multiple representation-data-digest values from which the receiver chooses. Endpoints are advised that sending multiple values consumes resources, which may be wasted if the receiver ignores them (see Section 3).

8.7. Duplicate digest-algorithm in field value

An endpoint might receive multiple representation-data-digest values (see Section 3) that use the same digest-algorithm with different or identical digest-values. For example:

```
Digest: sha-256=X48E9qOokqqrvdts8nOJRJN3OWDUoyWxBf7kbu9DBPE=,
        sha-256=47DEQpj8HBSa+/TImW+5JCeuQeRkm5NMpJWZG3hSuFU=
```

A receiver is permitted to ignore any representation-data-digest value, so validation of duplicates is left as an implementation decision. Endpoints might select all, some, or none of the values for checksum comparison and, based on the intersection of those results, conditionally pass or fail digest validation.

8.8. Resource exhaustion

Digest fields validation consumes computational resources. In order to avoid resource exhaustion, implementations can restrict validation of the algorithm types, number of validations, or the size of content.

9. IANA Considerations

9.1. Establish the HTTP Digest Algorithm Values Registry

This memo sets this specification to be the establishing document for the HTTP Digest Algorithm Values registry.

IANA is asked to update the "Reference" for this registry to refer this document and to inizialize the registry with the tokens defined in Section 6.

This registry uses the Specification Required policy (Section 4.6 of [RFC8126]).

9.2. Obsolete "contentMD5" token in Digest Algorithm

This memo adds the "contentMD5" token in the HTTP Digest Algorithm Values registry:

*Digest Algorithm: contentMD5

*Description: Section 5 of [RFC3230] defined the "contentMD5" token to be used only in Want-Digest. This token is obsoleted and MUST NOT be used.

*Reference: Section 9.2 of this document, Section 5 of [RFC3230].

*Status: obsoleted

9.3. Changes Compared to RFC3230

The contentMD5 digest-algorithm token defined in Section 5 of [RFC3230] has been added to the HTTP Digest Algorithm Values Registry with the "obsoleted" status.

All digest-algorithms defined in [RFC3230] are now "deprecated".

9.4. Changes Compared to RFC5843

The digest-algorithm tokens for "MD5", "SHA", "SHA-256", "SHA-512" have been updated to lowercase.

The status of "MD5" and "SHA" has been updated to "deprecated", and their description has been modified accordingly.

The "id-sha-256" and "id-sha-512" algorithms have been added to the registry.

9.5. Want-Digest Field Registration

This section registers the Want-Digest field in the "Hypertext Transfer Protocol (HTTP) Field Name Registry" [SEMANTICS].

Field name: Want-Digest

Status: permanent

Specification document(s): Section 5 of this document

9.6. Digest Field Registration

This section registers the Digest field in the "Hypertext Transfer Protocol (HTTP) Field Name Registry" [SEMANTICS].

Field name: Digest

Status: permanent

Specification document(s): Section 3 of this document

9.7. Want-Content-Digest Field Registration

This section registers the Want-Content-Digest field in the "Hypertext Transfer Protocol (HTTP) Field Name Registry" [SEMANTICS].

Field name: Want-Content-Digest

Status: permanent

Specification document(s): Section 5 of this document

9.8. Content-Digest Field Registration

This section registers the Content-Digest field in the "Hypertext Transfer Protocol (HTTP) Field Name Registry" [SEMANTICS].

Field name: Content-Digest

Status: permanent

Specification document(s): Section 4 of this document

10. References

10.1. Normative References

[CMU-836068] Carnagie Mellon University, Software Engineering Institute, "MD5 Vulnerable to collision attacks", 31 December 2008, <https://www.kb.cert.org/vuls/id/836068/>.

[IACR-2020-014] Leurent, G. and T. Peyrin, "SHA-1 is a Shambles", 5 January 2020, <https://eprint.iacr.org/2020/014.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/rfc/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/rfc/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/rfc/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/rfc/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/rfc/rfc3230>.

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

[RFC4960] Stewart, R., Ed., "Stream Control Transmission Protocol", RFC 4960, DOI 10.17487/RFC4960, September 2007, <https://www.rfc-editor.org/rfc/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/rfc/rfc5234>.

[RFC5843] Bryan, A., "Additional Hash Algorithms for HTTP Instance Digests", RFC 5843, DOI 10.17487/RFC5843, April 2010, <https://www.rfc-editor.org/rfc/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/rfc/rfc6234>.

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

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/rfc/rfc8126>.

[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/rfc/rfc8174>.

[SEMANTICS] Fielding, R. T., Nottingham, M., and J. Reschke, "HTTP Semantics", Work in Progress, Internet-Draft, draft-ietf-httpbis-semantics-19, 12 September 2021, <https://datatracker.ietf.org/doc/html/draft-ietf-httpbis-semantics-19>.

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

10.2. Informative References

[HTTP11] Fielding, R. T., Nottingham, M., and J. Reschke, "HTTP/1.1", Work in Progress, Internet-Draft, draft-ietf-httpbis-messaging-19, 12 September 2021, <https://datatracker.ietf.org/doc/html/draft-ietf-httpbis-messaging-19>.

[I-D.ietf-httpbis-header-structure] Nottingham, M. and P. Kamp, "Structured Field Values for HTTP", Work in Progress, Internet-Draft, draft-ietf-httpbis-header-structure-19, 3 June 2020, <https://datatracker.ietf.org/doc/html/draft-ietf-httpbis-header-structure-19>.

[I-D.thomson-http-mice] Thomson, M. and J. Yasskin, "Merkle Integrity Content Encoding", Work in Progress, Internet-Draft, draft-thomson-http-mice-03, 13 August 2018, <https://datatracker.ietf.org/doc/html/draft-thomson-http-mice-03>.

[NO-MD5] Turner, S. and L. Chen, "Updated Security Considerations for the MD5 Message-Digest and the HMAC-MD5 Algorithms", RFC 6151, DOI 10.17487/RFC6151, March 2011, <https://www.rfc-editor.org/rfc/rfc6151>.

[NO-SHA1] Polk, T., Chen, L., Turner, S., and P. Hoffman, "Security Considerations for the SHA-0 and SHA-1 Message-Digest Algorithms", RFC 6194, DOI 10.17487/RFC6194, March 2011, <https://www.rfc-editor.org/rfc/rfc6194>.

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

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

[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/rfc/rfc7231>.

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

[RFC7696] Housley, R., "Guidelines for Cryptographic Algorithm Agility and Selecting Mandatory-to-Implement Algorithms", BCP 201, RFC 7696, DOI 10.17487/RFC7696, November 2015, <https://www.rfc-editor.org/rfc/rfc7696>.

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

[RFC8188] Thomson, M., "Encrypted Content-Encoding for HTTP", RFC 8188, DOI 10.17487/RFC8188, June 2017, <https://www.rfc-editor.org/rfc/rfc8188>.

Appendix A. Resource Representation and Representation-Data

The following examples show how representation metadata, payload transformations and method impacts on the message and content. When the content contains non-printable characters (e.g. when it is compressed) it is shown as a Base64-encoded string.

```
PUT /entries/1234 HTTP/1.1
Host: foo.example
Content-Type: application/json

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

Figure 1: Request containing a JSON object without any content coding

```
PUT /entries/1234 HTTP/1.1
Host: foo.example
Content-Type: application/json
Content-Encoding: gzip

H4sIAItWyFwC/6tWSlSyUlAypANQqgUAREcqfG0AAAA=
```

Figure 2: Request containing a gzip-encoded JSON object

Now the same content conveys a malformed JSON object, because the request does not indicate a content coding.

```
PUT /entries/1234 HTTP/1.1
Host: foo.example
Content-Type: application/json

H4sIAItWyFwC/6tWSlSyUlAypANQqgUAREcqfG0AAAA=
```

Figure 3: Request containing malformed JSON

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

```
GET /entries/1234 HTTP/1.1
Host: foo.example
Range: bytes=1-7
```

Figure 4: Request for partial content

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

iwgAla3RXA==
```

Figure 5: Partial response from a gzip-encoded representation

The method can also alter the content. For example, the response to a HEAD request does not carry content.

```
HEAD /entries/1234 HTTP/1.1
Host: foo.example
Accept: application/json
Accept-Encoding: gzip
```

Figure 6: HEAD request

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

Figure 7: Response to HEAD request (empty content)

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, even though the request is directed to /authors/.

```
POST /authors/ HTTP/1.1
Host: foo.example
Accept: application/json
Content-Type: application/json

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

Figure 8: POST request

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

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

Figure 9: Response with Content-Location header

Appendix B. Examples of Unsolicited Digest

The following examples demonstrate interactions where a server responds with a Digest or Content-Digest fields even though the client did not solicit one using Want-Digest or Want-Content-Digest.

Some examples include JSON objects in the content. For presentation purposes, objects that fit completely within the line-length limits are presented on a single line using compact notation with no leading space. Objects that would exceed line-length limits are presented across multiple lines (one line per key-value pair) with 2 spaced of leading indentation.

Checksum mechanisms defined in this document are media-type agnostic and do not provide canonicalization algorithms for specific formats. Examples are calculated inclusive of any space. While examples can include both fields, Digest and Content-Digest can be returned independently.

B.1. Server Returns Full Representation Data

In this example, the message content conveys complete representation data, so Digest and Content-Digest have the same value.

```
GET /items/123 HTTP/1.1
Host: foo.example
```

Figure 10: GET request for an item

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

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

Figure 11: Response with Content-Digest

B.2. Server Returns No Representation Data

In this example, a HEAD request is used to retrieve the checksum of a resource.

The response Digest field-value is calculated over the JSON object {"hello": "world"}, which is not shown because there is no payload data. Content-Digest is computed on empty content.

```
HEAD /items/123 HTTP/1.1
Host: foo.example
```

Figure 12: HEAD request for an item

```
HTTP/1.1 200 OK
Content-Type: application/json
Digest: sha-256=X48E9qOokqqrvdts8nOJRJN3OWDUoyWxBf7kbu9DBPE=
Content-Digest: sha-256=47DEQpj8HBSa+/TImW+5JCeuQeRkm5NMpJWZG3hSuFU=
```

Figure 13: Response with both Content-Digest and Digest; empty content

B.3. Server Returns Partial Representation Data

In this example, the client makes a range request and the server responds with partial content. The Digest field-value represents the entire JSON object {"hello": "world"}, while the Content-Digest field-value is computed on the message content "hello".

```
GET /items/123 HTTP/1.1
Host: foo.example
Range: bytes=1-7
```

Figure 14: Request for partial content

```
HTTP/1.1 206 Partial Content
Content-Type: application/json
Content-Range: bytes 1-7/18
Digest: sha-256=X48E9qOokqqrvdts8nOJRJN3OWDUoyWxBf7kbu9DBPE=
Content-Digest: sha-256=Wqdirjg/u3J688ejbUlApbjECpiUUtIwT8lY/z81Tno=

"hello"
```

Figure 15: Partial response with both Content-Digest and Digest

B.4. Client and Server Provide Full Representation Data

The request contains a Digest field-value 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.

For presentation purposes, the response body is displayed as a Base64-encoded string because it contains non-printable characters.

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

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

Figure 16: PUT Request with Digest

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Location: /items/123
Content-Encoding: br
Content-Length: 22
Digest: sha-256=4REjxQ4yrqUVicfSKYNO/cF9zNj5ANbzgDZt3/h3Qxo=

iwiAeyJoZWxsbyI6ICJ3b3JsZCJ9Aw==
```

Figure 17: Response with Digest of encoded response

B.5. Client Provides Full Representation Data, Server Provides No Representation Data

The request Digest field-value is calculated on the enclosed payload.

The response Digest field-value depends on the representation metadata header fields, including Content-Encoding: br even when the response does not contain content.

```
PUT /items/123 HTTP/1.1
Host: foo.example
Content-Type: application/json
Content-Length: 18
Accept-Encoding: br
Digest: sha-256=X48E9qOokqqrvdts8nOJRJN3OWDUoyWxBf7kbu9DBPE=

{"hello": "world"}


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

Figure 18: Empty response with Digest

B.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.

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

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

Figure 19: PUT Request with Digest

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Encoding: br
Content-Location: /items/123
Digest: sha-256=4REjxQ4yrqUVicfSKYNO/cF9zNj5ANbzgDZt3/h3Qxo=,
        id-sha-256=X48E9qOokqqrvdts8nOJRJN3OWDUoyWxBf7kbu9DBPE=

iwiAeyJoZWxsbyI6ICJ3b3JsZCJ9Aw==
```

Figure 20: Response with Digest of Encoded Content

B.7. POST Response does not Reference the Request URI

The request Digest field-value is computed on the enclosed representation (see Section 7).

The representation enclosed in the response refers to the resource identified by Content-Location (see Section 6.4.2 of [SEMANTICS]). Digest is thus computed on the enclosed representation.

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

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

Figure 21: POST Request with Digest

```
HTTP/1.1 201 Created
Content-Type: application/json
Content-Location: /books/123
Location: /books/123
Digest: id-sha-256=yxOAqEeoj+reqygSIsLpT0LhumrNkIds5uLKtmdLyYE=

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

Figure 22: Response with Digest of Resource

Note that a 204 No Content response without content but with the same Digest field-value would have been legitimate too. In that case, Content-Digest would have been computed on an empty content.

B.8. POST Response Describes the Request Status

The request Digest field-value is computed on the enclosed representation (see Section 7).

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.

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

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

Figure 23: POST Request with Digest

```
HTTP/1.1 201 Created
Content-Type: application/json
Digest: id-sha-256=2LBp5RKZGpsSNf8BPXlXrX4Td4Tf5R5bZ9z7kdi5VvY=
Location: /books/123

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

Figure 24: Response with Digest of Representation

B.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 field-value is computed on the complete representation of the patched resource.

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

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

Figure 25: PATCH Request with Digest

```
HTTP/1.1 200 OK
Content-Type: application/json
Digest: id-sha-256=yxOAqEeoj+reqygSIsLpT0LhumrNkIds5uLKtmdLyYE=

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

Figure 26: Response with Digest of Representation

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

B.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 sends the same request from Figure 25 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 response Digest field-value is computed on this enclosed representation.

```
HTTP/1.1 404 Not Found
Content-Type: application/problem+json
Digest: sha-256=KPqhVXAT25LLitV1w0O167unHmVQusu+fpxm65zAsvk=

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

Figure 27: Response with Digest of Error Representation

B.11. Use with Trailer Fields and Transfer Coding

An origin server sends Digest as trailer field, so it can calculate digest-value while streaming content and thus mitigate resource consumption. The Digest field-value is the same as in Appendix B.1 because Digest is designed to be independent from the use of one or more transfer codings (see Section 2).

```
GET /items/123 HTTP/1.1
Host: foo.example
```

Figure 28: GET Request

```
HTTP/1.1 200 OK
Content-Type: application/json
Transfer-Encoding: chunked
Trailer: Digest

8\r\n
{"hello"\r\n
8
: "world\r\n
2\r\n
"}\r\n
0\r\n
Digest: sha-256=X48E9qOokqqrvdts8nOJRJN3OWDUoyWxBf7kbu9DBPE=
```

Figure 29: Chunked Response with Digest

Appendix C. Examples of Want-Digest Solicited Digest

The following examples demonstrate interactions where a client solicits a Digest using Want-Digest. The behavior of Content-Digest and Want-Content-Digest is identical.

Some examples include JSON objects in the content. For presentation purposes, objects that fit completely within the line-length limits are presented on a single line using compact notation with no leading space. Objects that would exceed line-length limits are presented across multiple lines (one line per key-value pair) with 2 spaced of leading indentation.

Checksum mechanisms described in this document are media-type agnostic and do not provide canonicalization algorithms for specific formats. Examples are calculated inclusive of any space.

C.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.

```
GET /items/123 HTTP/1.1
Host: foo.example
Want-Digest: sha-256;q=0.3, sha;q=1
```

Figure 30: GET Request with Want-Digest

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

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

Figure 31: Response with Different Algorithm

C.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.

```
GET /items/123 HTTP/1.1
Host: foo.example
Want-Digest: sha;q=1
```

Figure 32: GET Request with Want-Digest

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

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

Figure 33: Response with Unsupported Algorithm

C.3. Server Does Not Support Client Algorithm and Returns an Error

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

```
GET /items/123 HTTP/1.1
Host: foo.example
Want-Digest: sha;q=1
```

Figure 34: GET Request with Want-Digest

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

Figure 35: Response with Want-Digest

Appendix D. Changes from RFC3230

D.1. Deprecate Negotiation of Content-MD5

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

See Section 4 for a new checksum negotiation mechanism for HTTP message content.

D.2. Obsolete Digest Field Parameters

Sections 4.1.1 and 4.2 of [RFC3230] defined field parameters. This document obsoletes the usage of parameters with Digest because this feature has not been widely deployed and complicates field-value processing.

[RFC3230] intended field parameters to provide a common way to attach additional information to a representation-data-digest. However, if parameters are used as an input to validate the checksum, an attacker could alter them to steer the validation behavior.

A digest-algorithm can still be parameterized by defining its own way to encode parameters into the representation-data-digest, in such a way as to mitigate security risks related to its computation.

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.

Thanks to Julian Reschke for his valuable contributions to this document, and to the following contributors that have helped improve

this specification by reporting bugs, asking smart questions, drafting or reviewing text, and evaluating open issues: Mike Bishop, Brian Campbell, Matthew Kerwin, James Manger, Tommy Pauly, Sean Turner, and Erik Wilde.

FAQ

RFC Editor: Please remove this section before publication.

1. Why remove all references to content-md5?

   Those were unnecessary to understanding and using this specification.

2. Why remove references to instance manipulation?

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

3. How to use Digest with PATCH method?

   See Section 7.

4. Why remove references to delta-encoding?

   Unnecessary for a correct implementation of this specification. The revised specification 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 specification seems to me completely unrelated to Digest Authentication but for the word "Digest".

6. What changes in Want-Digest?

   The contentMD5 token defined in Section 5 of [RFC3230] is deprecated by Appendix D.1.

   To clarify that Digest and Want-Digest can be used in both requests and responses - [RFC3230] carefully uses sender and receiver in their definition - we added examples on using Want-Digest in responses to advertise the supported digest-algorithms and the inability to accept requests with unsupported digest-algorithms.

7. Does this specification change supported algorithms?

   Yes. 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. To simplify a future transition to Structured Fields [I-D.ietf-httpbis-header-structure] we suggest to use lowercase for digest-algorithms.

8. What about mid-stream trailer fields?

   While mid-stream trailer fields are interesting, since this specification is a rewrite of [RFC3230] we do not think we should face that. As a first thought, nothing in this document precludes future work that would find a use for mid-stream trailers, for example an incremental digest-algorithm. A document defining such a digest-algorithm is best positioned to describe how it is used.

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, logging
log = logging.getLogger()

def encode_item(item, encoding=lambda x: x):
    indent = 2 if isinstance(item, dict) and len(item) > 1 else None
    json_bytes = json.dumps(item, indent=indent).encode()
    return encoding(json_bytes)


def digest_bytes(bytes_, algorithm=hashlib.sha256):
    checksum_bytes = algorithm(bytes_).digest()
    log.warning("Log bytes: \n[%r]", bytes_)
    return base64.encodebytes(checksum_bytes).strip()


def digest(item, encoding=lambda x: x, algorithm=hashlib.sha256):
    content_encoded = encode_item(item, encoding)
    return digest_bytes(content_encoded, algorithm)


item = {"hello": "world"}

print("Encoding | digest-algorithm | digest-value")
print("Identity | sha256 |", digest(item))
# Encoding | digest-algorithm | digest-value
# Identity | sha256 | X48E9qOokqqrvdts8nOJRJN3OWDUoyWxBf7kbu9DBPE=

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/7XgHaAy8pmojAkGWoRx2UFChF41A2svX+TaPm'
#                      '+AbwAgBWnrIiYllu7BNNyealdVLvRwE\nmTHWXvJwew=='
```

Changes

RFC Editor: Please remove this section before publication.

Since draft-ietf-httpbis-digest-headers-05

- *Reboot digest-algorithm values registry #1567

- *Add Content-Digest #1542

- *Remove SRI section #1478

Since draft-ietf-httpbis-digest-headers-04

*Improve SRI section #1354

*About duplicate digest-algorithms #1221

*Improve security considerations #852

*md5 and sha deprecation references #1392

*Obsolete 3230 #1395

*Editorial #1362

Since draft-ietf-httpbis-digest-headers-03

*Reference semantics-12

*Detail encryption quirks

*Details on Algorithm agility #1250

*Obsolete parameters #850

Since draft-ietf-httpbis-digest-headers-02

*Deprecate SHA-1 #1154

Avoid id- with encrypted content

*Digest is independent from MESSAGING and HTTP/1.1 is not normative #1215

*Identity is not a valid field value for content-encoding #1223

*Mention trailers #1157

*Reference httpbis-semantics #1156

*Add contentMD5 as an obsoleted digest-algorithm #1249

*Use lowercase digest-algorithms names in the doc and in the digest-algorithm IANA table.

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

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

Authors' Addresses

Roberto Polli
Team Digitale, Italian Government
Italy

Email: robipolli@gmail.com

Lucas Pardue
Cloudflare

Email: lucaspardue.24.7@gmail.com