Workgroup: HTTP
Internet-Draft:
draft-ietf-httpbis-digest-headers-13
Obsoletes: 3230 (if approved)
Published: 10 July 2023
Intended Status: Standards Track
Expires: 11 January 2024
Authors: R. Polli — Team Digitale, Italian Government
L. Pardue — Cloudflare

Digest Fields

Abstract

This document defines HTTP fields that support integrity digests. The Content-Digest field can be used for the integrity of HTTP message content. The Repr-Digest field can be used for the integrity of HTTP representations. Want-Content-Digest and Want-Repr-Digest can be used to indicate a sender's interest and preferences for receiving the respective Integrity fields.

This document obsoletes RFC 3230 and the Digest and Want-Digest HTTP fields.

About This Document

This note is to be removed before publishing as an RFC.

Status information for this document may be found at https://datatracker.ietf.org/doc/draft-ietf-httpbis-digest-headers/.

Discussion of this document takes place on the HTTP Working Group mailing list (mailto:ietf-http-wg@w3.org), which is archived at https://lists.w3.org/Archives/Public/ietf-http-wg/. Working Group information can be found at https://httpwg.org/.

Source for this draft and an issue tracker can be found at https://github.com/httpwg/http-extensions/labels/digest-headers.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction
  1.1. Document Structure
  1.2. Concept Overview
  1.3. Obsoleting RFC 3230
  1.4. Notational Conventions
2. The Content-Digest Field
3. The Repr-Digest Field
  3.1. Using Repr-Digest in State-Changing Requests
  3.2. Repr-Digest and Content-Location in Responses
4. Integrity preference fields
5. Hash Algorithm Considerations and Registration
6. Security Considerations
  6.1. HTTP Messages Are Not Protected In Full
  6.2. End-to-End Integrity
  6.3. Usage in Signatures
  6.4. Usage in Trailer Fields
  6.5. Variations Within Content Encoding
  6.6. Algorithm Agility
  6.7. Resource exhaustion
7. IANA Considerations
  7.1. HTTP Field Name Registration
  7.2. Establish the Hash Algorithms for HTTP Digest Fields Registry
  7.3. Deprecate the Hypertext Transfer Protocol (HTTP) Digest Algorithm Values Registry

8. References
  8.1. Normative References
  8.2. Informative References
Appendix A. Resource Representation and Representation Data
Appendix B. Examples of Unsolicited Digest
  B.1. Server Returns Full Representation Data
  B.2. Server Returns No Representation Data
  B.3. Server Returns Partial Representation Data
  B.4. Client and Server Provide Full Representation Data
  B.5. Client Provides Full Representation Data, Server Provides No Representation Data
  B.6. Client and Server Provide Full Representation Data
  B.7. POST Response does not Reference the Request URI
  B.8. POST Response Describes the Request Status
  B.9. Digest with PATCH
  B.10. Error responses
  B.11. Use with Trailer Fields and Transfer Coding
Appendix C. Examples of Want-Repr-Digest Solicited Digest
  C.1. Server Selects Client's Least Preferred Algorithm
  C.2. Server Selects Algorithm Unsupported by Client
  C.3. Server Does Not Support Client Algorithm and Returns an Error
Appendix D. Sample Digest Values
Appendix E. Migrating from RFC 3230
Acknowledgements
Code Samples
Changes
  Since draft-ietf-httpbis-digest-headers-12
  Since draft-ietf-httpbis-digest-headers-11
  Since draft-ietf-httpbis-digest-headers-10
  Since draft-ietf-httpbis-digest-headers-09
  Since draft-ietf-httpbis-digest-headers-08
  Since draft-ietf-httpbis-digest-headers-07
  Since draft-ietf-httpbis-digest-headers-06
  Since draft-ietf-httpbis-digest-headers-05
  Since draft-ietf-httpbis-digest-headers-04
  Since draft-ietf-httpbis-digest-headers-03
  Since draft-ietf-httpbis-digest-headers-02
  Since draft-ietf-httpbis-digest-headers-01
  Since draft-ietf-httpbis-digest-headers-00
Authors' Addresses

1. Introduction

HTTP does not define the means to protect the data integrity of content or representations. When HTTP messages are transferred between endpoints, lower layer features or properties such as TCP checksums or TLS records [TLS] can provide some integrity protection. However, transport-oriented integrity provides a limited utility because it is opaque to the application layer and only covers the extent of a single connection. HTTP messages often travel over a chain of separate connections. In between connections there is a possibility for data corruption. An HTTP integrity mechanism can provide the means for endpoints, or applications using HTTP, to detect data corruption and make a choice about how to act on it. An example use case is to aid fault detection and diagnosis across system boundaries.

This document defines two digest integrity mechanisms for HTTP. First, content integrity, which acts on conveyed content (Section 6.4 of [HTTP]). Second, representation data integrity, which acts on representation data (Section 8.1 of [HTTP]). This supports advanced use cases such as validating the integrity of a resource that was reconstructed from parts retrieved using multiple requests or connections.

This document obsoletes RFC 3230 and therefore the Digest and Want-Digest HTTP fields; see Section 1.3.

1.1. Document Structure

This document is structured as follows:

* New request and response header and trailer field definitions.
  - Section 2 (Content-Digest),
  - Section 3 (Repr-Digest), and
  - Section 4 (Want-Content-Digest and Want-Repr-Digest).
* Considerations specific to representation data integrity.
  - Section 3.1 (State-changing requests),
  - Section 3.2 (Content-Location),
  - Appendix A contains worked examples of Representation data in message exchanges, and
  - Appendix B and Appendix C contain worked examples of Repr-Digest and Want-Repr-Digest fields in message exchanges.
* Section 5 presents hash algorithm considerations and defines registration procedures for future entries.

1.2. Concept Overview

The HTTP fields defined in this document can be used for HTTP integrity. Senders choose a hashing algorithm and calculate a digest from an input related to the HTTP message. The algorithm identifier and digest are transmitted in an HTTP field. Receivers can validate the digest for integrity purposes. Hashing algorithms are registered in the "Hash Algorithms for HTTP Digest Fields" registry (see Section 7.2).

Selecting the data on which digests are calculated depends on the use case of the HTTP messages. This document provides different fields for HTTP representation data and HTTP content.

There are use cases where a simple digest of the HTTP content bytes is required. The Content-Digest request and response header and trailer field is defined to support digests of content (Section 6.4 of [HTTP]); see Section 2.

For more advanced use cases, the Repr-Digest request and response header and trailer field (Section 3) is defined. It contains a digest value computed by applying a hashing algorithm to selected representation data (Section 8.1 of [HTTP]). Basing Repr-Digest on the selected representation makes it straightforward to apply it to use cases where the message content requires some sort of manipulation to be considered as representation of the resource or content conveys a partial representation of a resource, such as Range Requests (see Section 14 of [HTTP]).

Content-Digest and Repr-Digest support hashing algorithm agility. The Want-Content-Digest and Want-Repr-Digest fields allow endpoints to express interest in Content-Digest and Repr-Digest respectively, and to express algorithm preferences in either.

Content-Digest and Repr-Digest are collectively termed Integrity fields. Want-Content-Digest and Want-Repr-Digest are collectively termed Integrity preference fields.

Integrity fields are tied to the Content-Encoding and Content-Type header fields. Therefore, a given resource may have multiple different digest values when transferred with HTTP.

Integrity fields apply to HTTP message content or HTTP representations. They do not apply to 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. For example, HTTP Message Signatures [SIGNATURES] could be used to sign Integrity fields, thus providing coverage for HTTP content or representation data.

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

1.3. Obsoleting RFC 3230

[RFC3230] defined the Digest and Want-Digest HTTP fields for HTTP integrity. It also coined the term "instance" and "instance manipulation" in order to explain concepts that are now more universally defined, and implemented, as HTTP semantics such as selected representation data (Section 8.1 of [HTTP]).

Experience has shown that implementations of [RFC3230] have interpreted the meaning of "instance" inconsistently, leading to interoperability issues. The most common issue relates to the mistake of calculating the digest using (what we now call) message content, rather than using (what we now call) representation data as was originally intended. Interestingly, time has also shown that a digest of message content can be beneficial for some use cases. So it is difficult to detect if non-conformance to [RFC3230] is intentional or unintentional.

In order to address potential inconsistencies and ambiguity across implementations of Digest and Want-Digest, this document obsoletes [RFC3230]. The Integrity fields (Sections 2 and 3) and Integrity preference fields (Section 4) defined in this document are better aligned with current HTTP semantics and have names that more clearly articulate the intended usages.

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]. This includes the rules: CR (carriage return), LF (line feed), and CRLF (CR LF).

This document uses the following terminology from Section 3 of [STRUCTURED-FIELDS] to specify syntax and parsing: Boolean, Byte Sequence, Dictionary, Integer, and List.

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

This document uses the line folding strategies described in [FOLDING].

Hashing algorithm names respect the casing used in their definition document (e.g., SHA-1, CRC32c).

HTTP messages indicate hashing algorithms using an Algorithm Key (algorithms). Where the document refers to an Algorithm Key in prose, it is quoted (e.g., "sha", "crc32c").

The term "checksum" describes the output of the application of an algorithm to a sequence of bytes, whereas "digest" is only used in relation to the value contained in the fields.

Integrity fields: collective term for Content-Digest and Repr-Digest

Integrity preference fields: collective term for Want-Repr-Digest and Want-Content-Digest

2. The Content-Digest Field

The Content-Digest HTTP field can be used in requests and responses to communicate digests that are calculated using a hashing algorithm applied to the actual message content (see Section 6.4 of [HTTP]). It is a Dictionary (see Section 3.2 of [STRUCTURED-FIELDS]) where each:

- key conveys the hashing algorithm (see Section 5) used to compute the digest;
- value is a Byte Sequence (Section 3.3.5 of [STRUCTURED-FIELDS]), that conveys an encoded version of the byte output produced by the digest calculation.

For example:

```
NOTE: '\' line wrapping per RFC 8792

Content-Digest: \
  sha-512=:YMAam51Jz/jOATT6/zvHrLVgOYTGFy1d6GJiOHTohq4yP+pgk4vf2aCs\
  yRZOtw8MjkM7iw7yZ/WkppmM44T3qg==:
```

The Dictionary type can be used, for example, to attach multiple digests calculated using different hashing algorithms in order to support a population of endpoints with different or evolving capabilities. Such an approach could support transitions away from weaker algorithms (see Section 6.6).

```
NOTE: '\' line wrapping per RFC 8792

Content-Digest: \
  sha-256=:d435Qo+nKZ+gLcUHn7GQtQ72hiBVAgqoLsZnZPiTGPk=:,\
  sha-512=:YMAam51Jz/jOATT6/zvHrLVgOYTGFy1d6GJiOHTohq4yP+pgk4vf2aCs\
  yRZOtw8MjkM7iw7yZ/WkppmM44T3qg==:
```

A recipient **MAY** ignore any or all digests. Application-specific behavior or local policy **MAY** set additional constraints on the processing and validation practices of the conveyed digests. The security considerations covers some of the issues related to ignoring digests (see Section 6.6) and validating multiple digests (see Section 6.7).

A sender **MAY** send a digest without knowing whether the recipient supports a given hashing 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 [HTTP].

3. The Repr-Digest Field

The Repr-Digest HTTP field can be used in requests and responses to communicate digests that are calculated using a hashing algorithm applied to the entire selected representation data (see Section 8.1 of [HTTP]).

Representations take into account the effect of the HTTP semantics on 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 [HTTP/1.1]). To help illustrate HTTP representation concepts, several examples are provided in Appendix A.

When a message has no representation data it is still possible to assert that no representation data was sent by computing the digest on an empty string (see Section 6.3).

Repr-Digest is a Dictionary (see Section 3.2 of [STRUCTURED-FIELDS]) where each:

* key conveys the hashing algorithm (see Section 5) used to compute the digest;

* value is a Byte Sequence, that conveys an encoded version of the byte output produced by the digest calculation.

For example:

```
NOTE: '\' line wrapping per RFC 8792

Repr-Digest: \
  sha-512=:YMAam51Jz/jOATT6/zvHrLVgOYTGFy1d6GJiOHTohq4yP+pgk4vf2aCs\
  yRZOtw8MjkM7iw7yZ/WkppmM44T3qg==:
```

The Dictionary type can be used, for example, to attach multiple digests calculated using different hashing algorithms in order to support a population of endpoints with different or evolving capabilities. Such an approach could support transitions away from weaker algorithms (see Section 6.6).

```
NOTE: '\' line wrapping per RFC 8792

Repr-Digest: \
  sha-256=:d435Qo+nKZ+gLcUHn7GQtQ72hiBVAgqoLsZnZPiTGPk=:,\
  sha-512=:YMAam51Jz/jOATT6/zvHrLVgOYTGFy1d6GJiOHTohq4yP+pgk4vf2aCs\
  yRZOtw8MjkM7iw7yZ/WkppmM44T3qg==:
```

A recipient **MAY** ignore any or all digests. Application-specific behavior or local policy **MAY** set additional constraints on the processing and validation practices of the conveyed digests. The security considerations covers some of the issues related to ignoring digests (see Section 6.6) and validating multiple digests (see Section 6.7).

A sender **MAY** send a digest without knowing whether the recipient supports a given hashing algorithm, or even knowing that the recipient will ignore it.

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

3.1. Using Repr-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 3).

In responses,

* if the representation describes the status of the request, Repr-Digest **MUST** be computed on the enclosed representation (see Appendix B.8);

* if there is a referenced resource Repr-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 Repr-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 [HTTP]). In contrast, the Location header field does not affect Repr-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.

3.2. Repr-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 Repr-Digest is computed accordingly. An example is given in Appendix B.7.

4. Integrity preference fields

Senders can indicate their interest in Integrity fields and hashing algorithm preferences using the Want-Content-Digest or Want-Repr-Digest fields. These can be used in both requests and responses.

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

Want-Repr-Digest indicates that the sender would like to receive a representation digest on messages associated with the request URI and representation metadata, using the Repr-Digest field.

If Want-Content-Digest or Want-Repr-Digest are used in a response, it indicates that the server would like the client to provide the respective Integrity field on future requests.

Integrity preference fields are only a hint. The receiver of the field can ignore it and send an Integrity field using any algorithm or omit the field entirely, for example see Appendix C.2. It is not a protocol error if preferences are ignored. Applications that use Integrity fields and Integrity preferences can define expectations or constraints that operate in addition to this specification. Ignored preferences are an application-specific concern.

Want-Content-Digest and Want-Repr-Digest are of type Dictionary where each:

* key conveys the hashing algorithm (see Section 5);

* value is an Integer (Section 3.3.1 of [STRUCTURED-FIELDS]) that conveys an ascending, relative, weighted preference. It must be in the range 0 to 10 inclusive. 1 is the least preferred, 10 is the most preferred, and a value of 0 means "not acceptable".

Examples:

```
Want-Repr-Digest: sha-256=1
Want-Repr-Digest: sha-512=3, sha-256=10, unixsum=0
Want-Content-Digest: sha-256=1
Want-Content-Digest: sha-512=3, sha-256=10, unixsum=0
```

5. Hash Algorithm Considerations and Registration

There are a wide variety of hashing algorithms that can be used for the purposes of integrity. The choice of algorithm depends on several factors such as the integrity use case, implementation needs or constraints, or application design and workflows.

An initial set of algorithms will be registered with IANA in the "Hash Algorithms for HTTP Digest Fields" registry; see Section 7.2. Additional algorithms can be registered in accordance with the policies set out in this section.

Each algorithm has a status field, which is intended to provide an aid to implementation selection.

Algorithms with a status value of "Active" are suitable for many purposes and it is **RECOMMENDED** that applications use these algorithms. These can be used in adversarial situations where hash functions might need to provide resistance to collision, first-preimage and second-preimage attacks. For adversarial situations, selecting which of the "Active" algorithms are acceptable will depend on the level of protection the circumstances demand. More considerations are presented in Section 6.6.

Algorithms with a status value of "Deprecated" either provide none of these properties, or are known to be weak (see [NO-MD5] and [NO-SHA]). These algorithms **MAY** be used to preserve integrity against corruption, but **MUST NOT** be used in a potentially adversarial setting; for example, when signing Integrity fields' values for authenticity. Permitting the use of these algorithms can help some applications, for example, those that previously used [RFC3230], are migrating to this specification (Appendix E), and have existing stored collections of computed digest values avoid undue operational overhead caused by recomputation using other more-secure algorithms. Such applications are not exempt from the requirements in this section. Furthermore, applications without such legacy or history ought to follow the guidance for using algorithms with the status value "Active".

Discussion of algorithm agility is presented in Section 6.6.

Registration requests for the "Hash Algorithms for HTTP Digest Fields" registry use the Specification Required policy (Section 4.6 of [RFC8126]). Requests should use the following template:

* Algorithm Key: the Structured Fields key value used in Content-Digest, Repr-Digest, Want-Content-Digest, or Want-Repr-Digest field Dictionary member keys

* Status: the status of the algorithm. The options are:

  - "Active" - for algorithms without known problems,

  - "Provisional" - for unproven algorithms,

  - "Deprecated" - for deprecated or insecure algorithms,

* Description: a short description of the algorithm

* Reference(s): pointer(s) to the primary document(s) defining the Algorithm Key and technical details of the algorithm

When reviewing registration requests, the designated expert(s) should pay attention to the requested status. The status value should reflect standardization status and the broad opinion of relevant interest groups such as the IETF or security-related SDOs. The "Active" status is not suitable for an algorithm that is known to be weak, broken, or experimental. If a registration request attempts to register such an algorithm as "Active", the designated expert(s) should suggest an alternative status of "Deprecated" or "Provisional".

When reviewing registration requests, the designated expert(s) cannot use a status of "Deprecated" or "Provisional" as grounds for rejection.

Requests to update or change the fields in an existing registration are permitted. For example, this could allow for the transition of an algorithm status from "Active" to "Deprecated" as the security environment evolves.

6. Security Considerations

6.1. HTTP Messages Are Not Protected In Full

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 corruption.

Integrity fields are not intended to be a general protection against malicious tampering with HTTP messages. In the absence of additional security mechanisms, an on-path, malicious actor can remove or recalculate and substitute a digest value. This attack can be mitigated by combining mechanisms described in this document with other approaches such as transport-layer security or digital signatures (for example, HTTP Message Signatures [SIGNATURES]).

6.2. End-to-End Integrity

Integrity fields can help detect representation data or content modification due to implementation errors, undesired "transforming proxies" (see Section 7.7 of [HTTP]) 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 a user agent can validate that resource retrieval succeeded before handing off to an HTML parser, video player, etc. for parsing.

Note that using these mechanisms 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 6.3.

6.3. 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 Integrity fields are included in this set.

There are no restrictions placed on the type or format of digital signature that Integrity fields can be used with. One possible approach is to combine them with HTTP Message Signatures [SIGNATURES].

Digests explicitly depend on the "representation metadata" (e.g., the values of Content-Type, Content-Encoding etc.). A signature that protects Integrity fields 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 3.2.

Signatures are likely to be deemed an adversarial setting when applying Integrity fields; see Section 5. Repr-Digest offers an interesting possibility when combined with signatures. In the scenario where there is no content to send, the digest of an empty string can be included in the message and, if signed, can help the recipient detect if content was added either as a result of accident or purposeful manipulation. The opposite scenario is also supported; including an Integrity field for content, and signing it, can help a recipient detect where the content was removed.

Any mangling of Integrity fields, including digests' de-duplication or combining different field values (see Section 5.2 of [HTTP]) might affect signature validation.

6.4. Usage in Trailer Fields

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

When Integrity 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.

One of the benefits of using Integrity fields in a trailer section is that it allows hashing of bytes as they are sent. However, it is possible to design a hashing algorithm that requires processing of content in such a way that would negate these benefits. For example, Merkle Integrity Content Encoding [I-D.thomson-http-mice] requires content to be processed in reverse order. This means the complete data needs to be available, which means there is negligible processing difference in sending an Integrity field in a header or trailer section.

6.5. Variations Within Content Encoding

Content coding mechanisms can support different encoding parameters, meaning that the same input content can produce different outputs. For example, GZIP supports multiple compression levels. Such encoding parameters are generally not communicated as representation metadata. For instance, different compression levels would all use the same "Content-Encoding: gzip" field. Other examples include where encoding relies on nonces or timestamps, such as the aes128gcm content coding defined in [RFC8188].

Since it is possible for there to be variation within content coding, the checksum conveyed by the integrity fields cannot be used to provide a proof of integrity "at rest" unless the whole content is persisted.

6.6. Algorithm Agility

The security properties of hashing algorithms are not fixed. Algorithm Agility (see [RFC7696]) is achieved by providing implementations with flexibility to choose hashing algorithms from the IANA Hash Algorithms for HTTP Digest Fields registry; see Section 7.2.

Transition from weak algorithms is supported by negotiation of hashing algorithm using Want-Content-Digest or Want-Repr-Digest (see Section 4) or by sending multiple digests from which the receiver chooses. A receiver that depends on a digest for security will be vulnerable to attacks on the weakest algorithm it is willing to accept. Endpoints are advised that sending multiple values consumes resources, which may be wasted if the receiver ignores them (see Section 3).

While algorithm agility allows the migration to stronger algorithms it does not prevent the use of weaker algorithms. Integrity fields do not provide any mitigations for downgrade or substitution attacks (see Section 1 of [RFC6211]) of the hashing algorithm. To protect against such attacks, endpoints could restrict their set of supported algorithms to stronger ones and protect the fields value by using TLS and/or digital signatures.

6.7. Resource exhaustion

Integrity 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. In these cases, skipping validation entirely or ignoring validation failure of a more-preferred algorithm leaves the possibility of a downgrade attack (see Section 6.6).

7. IANA Considerations

7.1. HTTP Field Name Registration

IANA is asked to update the "Hypertext Transfer Protocol (HTTP) Field Name Registry" registry ([HTTP]) according to the table below:

Field Name	Status	Reference
Content-Digest	permanent	Section 2 of this document
Repr-Digest	permanent	Section 3 of this document
	permanent	Section 4 of this document

Field Name	Status	Reference
Want-Content-Digest		
Want-Repr-Digest	permanent	Section 4 of this document
Digest	obsoleted	[RFC3230], Section 1.3 of this document
Want-Digest	obsoleted	[RFC3230], Section 1.3 of this document

Table 1

7.2. Establish the Hash Algorithms for HTTP Digest Fields Registry

IANA is requested to create the new "Hash Algorithms for HTTP Digest Fields" registry at https://www.iana.org/assignments/http-digest-hash-alg/ and populate it with the entries in Table 2. The procedure for new registrations is provided in Section 5.

Algorithm Key	Status	Description	Reference(s)
sha-512	Active	The SHA-512 algorithm.	[RFC6234], [RFC4648], this document.
sha-256	Active	The SHA-256 algorithm.	[RFC6234], [RFC4648], this document.
md5	Deprecated	The MD5 algorithm. It is vulnerable to collision attacks; see [NO-MD5] and [CMU-836068]	[RFC1321], [RFC4648], this document.
sha	Deprecated	The SHA-1 algorithm. It is vulnerable to collision attacks; see [NO-SHA] and [IACR-2020-014]	[RFC3174], [RFC4648], [RFC6234] this document.
unixsum	Deprecated	The algorithm used by the UNIX "sum" command.	[RFC4648], [RFC6234], [UNIX], this document.
unixcksum	Deprecated	The algorithm used by the UNIX "cksum" command.	[RFC4648], [RFC6234], [UNIX], this document.
adler	Deprecated	The ADLER32 algorithm.	[RFC1950], this document.
crc32c	Deprecated	The CRC32c algorithm.	[RFC9260] appendix A, this document.

Table 2: Initial Hash Algorithms

7.3. Deprecate the Hypertext Transfer Protocol (HTTP) Digest Algorithm Values Registry

IANA is requested to deprecate the "Hypertext Transfer Protocol (HTTP) Digest Algorithm Values" registry at https://www.iana.org/assignments/http-dig-alg/http-dig-alg.xhtml and replace the note on this registry with the following text:

> "This registry is deprecated since it lists the algorithms that can be used with the Digest and Want-Digest fields defined in [RFC3230]https://www.iana.org/, which has been obsoleted by [rfc-to-be-this-document]. While registration is not closed, new registrations are encouraged to use the [Hash Algorithms for HTTP Digest Fields]https://www.iana.org/assignments/http-digest-hash-alg/ registry instead.

8. References

8.1. Normative References

[FOLDING] Watsen, K., Auerswald, E., Farrel, A., and Q. Wu, "Handling Long Lines in Content of Internet-Drafts and RFCs", RFC 8792, DOI 10.17487/RFC8792, June 2020, <https://www.rfc-editor.org/rfc/rfc8792>.

[HTTP] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Semantics", STD 97, RFC 9110, DOI 10.17487/RFC9110, June 2022, <https://www.rfc-editor.org/rfc/rfc9110>.

[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. 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>.

[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>.

[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>.

[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>.

[STRUCTURED-FIELDS] Nottingham, M. and P. Kamp, "Structured Field Values for HTTP", RFC 8941, DOI 10.17487/RFC8941, February 2021, <https://www.rfc-editor.org/rfc/rfc8941>.

8.2. Informative References

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

[HTTP/1.1] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP/1.1", STD 99, RFC 9112, DOI 10.17487/RFC9112, June 2022, <https://www.rfc-editor.org/rfc/rfc9112>.

[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>.

[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>.

[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-SHA] 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>.

[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>.

[RFC6211] Schaad, J., "Cryptographic Message Syntax (CMS) Algorithm Identifier Protection Attribute", RFC 6211, DOI 10.17487/RFC6211, April 2011, <https://www.rfc-editor.org/rfc/rfc6211>.

[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>.

[RFC9260] Stewart, R., Tüxen, M., and K. Nielsen, "Stream Control Transmission Protocol", RFC 9260, DOI 10.17487/RFC9260, June 2022, <https://www.rfc-editor.org/rfc/rfc9260>.

[SIGNATURES] Backman, A., Richer, J., and M. Sporny, "HTTP Message Signatures", Work in Progress, Internet-Draft, draft-ietf-httpbis-message-signatures-17, 2 May 2023, <https://datatracker.ietf.org/doc/html/draft-ietf-httpbis-message-signatures-17>.

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

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

Appendix A. Resource Representation and Representation Data

This section following examples show how representation metadata, content transformations, and method impacts on the message and content. These examples a not exhaustive.

Unless otherwise indicated, the examples are based on the JSON object {"hello": "world"} followed by an LF. When the content contains non-printable characters (e.g., when it is encoded) it is shown as a sequence of hex-encoded bytes.

Consider a client that wishes to upload a JSON object using the PUT method. It could do this using the application/json content type without any content coding.

```
PUT /entries/1234 HTTP/1.1
Host: foo.example
Content-Type: application/json
Content-Length: 19

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

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

However, the use of content coding is quite common. The client could also upload the same data with a gzip coding (Section 8.4.1.3 of [HTTP]). Note that in this case, the Content-Length contains a larger value due to the coding overheads.

```
PUT /entries/1234 HTTP/1.1
Host: foo.example
Content-Type: application/json
Content-Encoding: gzip
Content-Length: 39

1F 8B 08 00 88 41 37 64 00 FF
AB 56 CA 48 CD C9 C9 57 B2 52
50 2A CF 2F CA 49 51 AA E5 02
00 D9 E4 31 E7 13 00 00 00
```

Figure 2: Request containing a gzip-encoded JSON object

Sending the gzip coded data without indicating it via Content-Encoding means that the content is malformed. In this case, the server can reply with an error.

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

Content-Length: 39

1F 8B 08 00 88 41 37 64 00 FF
AB 56 CA 48 CD C9 C9 57 B2 52
50 2A CF 2F CA 49 51 AA E5 02
00 D9 E4 31 E7 13 00 00 00
```

Figure 3: Request containing malformed JSON

```
HTTP/1.1 400 Bad Request
```

Figure 4: An error response for a malformed content

A Range-Request affects the transferred message content. In this example, the client is accessing the resource at /entries/1234, which is the JSON object {"hello": "world"} followed by an LF. However, the client has indicated a preferred content coding and a specific byte range.

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

Figure 5: Request for partial content

The server satisfies the client request by responding with a partial representation (equivalent to the first 10 of the JSON object displayed in whole in Figure 2).

```
HTTP/1.1 206 Partial Content
Content-Encoding: gzip
Content-Type: application/json
Content-Range: bytes 0-9/39

1F 8B 08 00 A5 B4 BD 62 02 FF
```

Figure 6: Partial response from a gzip-encoded representation

Aside from content coding or range requests, the method can also affect the transferred message content. For example, the response to a HEAD request does not carry content but in this example case does include a Content-Length; see Section 8.6 of [HTTP].

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

Figure 7: HEAD request

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Encoding: gzip
Content-Length: 39
```

Figure 8: Response to HEAD request (empty content)

Finally, the semantics of a response might decouple the target URI from the enclosed representation. In the example below, the client issues a POST request directed to /authors/ but the response includes a Content-Location header field that indicates the enclosed representation refers to the resource available at /authors/123. Note that Content-Length is not sent in this example.

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

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

Figure 9: POST request

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

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

Figure 10: Response with Content-Location header

Appendix B. Examples of Unsolicited Digest

The following examples demonstrate interactions where a server responds with a Content-Digest or Repr-Digest fields even though the client did not solicit one using Want-Content-Digest or Want-Repr-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 spaces 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, Content-Digest and Repr-Digest can be returned independently.

B.1. Server Returns Full Representation Data

In this example, the message content conveys complete representation data. This means that in the response, Content-Digest and Repr-Digest are both computed over the JSON object {"hello": "world"} followed by an LF, and thus have the same value.

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

Figure 11: GET request for an item

```
NOTE: '\' line wrapping per RFC 8792

HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 19
Content-Digest: \
  sha-256=:RK/0qy18MlBSVnWgjwz6lZEWjP/lF5HF9bvEF8FabDg=:
Repr-Digest: \
  sha-256=:RK/0qy18MlBSVnWgjwz6lZEWjP/lF5HF9bvEF8FabDg=:

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

Figure 12: Response with identical Repr-Digest and 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 Content-Digest field-value is computed on empty content. Repr-Digest is calculated over the JSON object {"hello": "world"} followed by an LF, which is not shown because there is no content.

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

Figure 13: HEAD request for an item

```
NOTE: '\' line wrapping per RFC 8792

HTTP/1.1 200 OK
Content-Type: application/json
Content-Digest: \
  sha-256=:47DEQpj8HBSa+/TImW+5JCeuQeRkm5NMpJWZG3hSuFU=:
Repr-Digest: \
  sha-256=:RK/0qy18MlBSVnWgjwz6lZEWjP/lF5HF9bvEF8FabDg=:
```

Figure 14: 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.

```
GET /items/123 HTTP/1.1
Host: foo.example
Range: bytes=10-18
```

Figure 15: Request for partial content

```
NOTE: '\' line wrapping per RFC 8792

HTTP/1.1 206 Partial Content
Content-Type: application/json
Content-Range: bytes 10-18/19
Content-Digest: \
  sha-256=:jjcgBDWNAtbYUXI37CVG3gRuGOAjaaDRGpIUFsdyepQ=:
Repr-Digest: \
  sha-256=:RK/0qy18MlBSVnWgjwz6lZEWjP/lF5HF9bvEF8FabDg=:

"world"}
```

Figure 16: Partial response with both Content-Digest and Repr-Digest

In the response message above, note that the Repr-Digest and Content-Digests are different. The Repr-Digest field-value is calculated across the entire JSON object {"hello": "world"} followed by an LF, and the field is

```
NOTE: '\' line wrapping per RFC 8792

Repr-Digest: \
  sha-256=:RK/0qy18MlBSVnWgjwz6lZEWjP/lF5HF9bvEF8FabDg=:
```

However, since the message content is constrained to bytes 10-18, the Content-Digest field-value is calculated over the sequence "world"} followed by an LF, thus resulting in

```
NOTE: '\' line wrapping per RFC 8792

Content-Digest: \
  sha-256=:jjcgBDWNAtbYUXI37CVG3gRuGOAjaaDRGpIUFsdyepQ=:
```

B.4. Client and Server Provide Full Representation Data

The request contains a Repr-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 Repr-Digest field-value is therefore different compared to the request.

For presentation purposes, the response body is displayed as a sequence of hex-encoded bytes because it contains non-printable characters.

```
NOTE: '\' line wrapping per RFC 8792

PUT /items/123 HTTP/1.1
Host: foo.example
Content-Type: application/json
Accept-Encoding: br
Repr-Digest: \
  sha-256=:RK/0qy18MlBSVnWgjwz6lZEWjP/lF5HF9bvEF8FabDg=:

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

Figure 17: PUT Request with Digest

```
NOTE: '\' line wrapping per RFC 8792

HTTP/1.1 200 OK
Content-Type: application/json
Content-Location: /items/123
Content-Encoding: br
Content-Length: 23
Repr-Digest: \
  sha-256=:d435Qo+nKZ+gLcUHn7GQtQ72hiBVAgqoLsZnZPiTGPk=:

8B 08 80 7B 22 68 65 6C 6C 6F
22 3A 20 22 77 6F 72 6C 64 22
7D 0A 03
```

Figure 18: Response with Digest of encoded response

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

The request Repr-Digest field-value is calculated on the enclosed content, which is the JSON object {"hello": "world"} followed by an LF

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

```
NOTE: '\' line wrapping per RFC 8792

PUT /items/123 HTTP/1.1
Host: foo.example
Content-Type: application/json
Content-Length: 19
Accept-Encoding: br
Repr-Digest: \
  sha-256=:RK/0qy18MlBSVnWgjwz6lZEWjP/lF5HF9bvEF8FabDg==:

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

```
HTTP/1.1 204 No Content
Content-Type: application/json
Content-Encoding: br
Repr-Digest: sha-256=:d435Qo+nKZ+gLcUHn7GQtQ72hiBVAgqoLsZnZPiTGPk=:
```

Figure 19: Empty response with Digest

B.6. Client and Server Provide Full Representation Data

The response contains two digest values using different algorithms.

For presentation purposes, the response body is displayed as a sequence of hex-encoded bytes because it contains non-printable characters.

```
NOTE: '\' line wrapping per RFC 8792

PUT /items/123 HTTP/1.1
Host: foo.example
Content-Type: application/json
Accept-Encoding: br
Repr-Digest: \
  sha-256=:RK/0qy18MlBSVnWgjwz6lZEWjP/lF5HF9bvEF8FabDg==:

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

Figure 20: PUT Request with Digest

```
NOTE: '\' line wrapping per RFC 8792

HTTP/1.1 200 OK
Content-Type: application/json
Content-Encoding: br
Content-Location: /items/123
Repr-Digest: \
  sha-256=:d435Qo+nKZ+gLcUHn7GQtQ72hiBVAgqoLsZnZPiTGPk=:,\
  sha-512=:db7fdBbgZMgX1Wb2MjA8zZj+rSNgfmDCEEXM8qLWfpfoNY0sCpHAzZbj\
  09X1/7HAb7Od5Qfto4QpuBsFbUO3dQ==:

8B 08 80 7B 22 68 65 6C 6C 6F
22 3A 20 22 77 6F 72 6C 64 22
7D 0A 03
```

Figure 21: Response with Digest of Encoded Content

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

The request Repr-Digest field-value is computed on the enclosed representation (see Section 3.1), which is the JSON object {"title": "New Title"} followed by an LF.

The representation enclosed in the response is a multiline JSON object followed by an LF. It refers to the resource identified by Content-Location (see Section 6.4.2 of [HTTP]); an application can thus use Repr-Digest in association with the resource referenced by Content-Location.

```
POST /books HTTP/1.1
Host: foo.example
Content-Type: application/json
Accept: application/json
Accept-Encoding: identity
Repr-Digest: sha-256=:mEkdbO7Srd9LIOegftO0aBX+VPTVz7/CSHes2Z27gc4=:

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

Figure 22: POST Request with Digest

```
HTTP/1.1 201 Created
Content-Type: application/json
Content-Location: /books/123
Location: /books/123
Repr-Digest: sha-256=:uVSlinTTdQUwm2On4k8TJUikGN1bf/Ds8WPX4oe0h9I=:

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

Figure 23: Response with Digest of Resource

B.8. POST Response Describes the Request Status

The request Repr-Digest field-value is computed on the enclosed representation (see Section 3.1), which is the JSON object {"title": "New Title"} followed by an LF.

The representation enclosed in the response describes the status of the request, so Repr-Digest is computed on that enclosed representation. It is a multiline JSON object followed by an LF.

Response Repr-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
Repr-Digest: sha-256=:mEkdbO7Srd9LIOegftO0aBX+VPTVz7/CSHes2Z27gc4=:

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

Figure 24: POST Request with Digest

```
HTTP/1.1 201 Created
Content-Type: application/json
Repr-Digest: sha-256=:yXIGDTN5VrfoyisKlXgRKUHHMs35SNtyC3szSz1dbO8=:
Location: /books/123

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

Figure 25: Response with Digest of Representation

B.9. Digest with PATCH

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

The PATCH request uses the application/merge-patch+json media type defined in [RFC7396]. Repr-Digest is calculated on the content, which corresponds to the patch document and is the JSON object {"title": "New Title"} followed by an LF.

The response Repr-Digest field-value is computed on the complete representation of the patched resource. It is a multiline JSON object followed by an LF.

```
PATCH /books/123 HTTP/1.1
Host: foo.example
Content-Type: application/merge-patch+json
Accept: application/json
Accept-Encoding: identity
Repr-Digest: sha-256=:mEkdbO7Srd9LIOegftO0aBX+VPTVz7/CSHes2Z27gc4=:

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

Figure 26: PATCH Request with Digest

```
HTTP/1.1 200 OK
Content-Type: application/json
Repr-Digest: sha-256=:uVSlinTTdQUwm2On4k8TJUikGN1bf/Ds8WPX4oe0h9I=:

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

Figure 27: Response with Digest of Representation

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

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 26 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 Repr-Digest field-value is computed on this enclosed representation. It is a multiline JSON object followed by an LF.

```
HTTP/1.1 404 Not Found
Content-Type: application/problem+json
Repr-Digest: sha-256=:EXB0S2VF2H7ijkAVJkH1Sm0pBho0iDZcvVUHHXTTZSA=:

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

Figure 28: Response with Digest of Error Representation

B.11. Use with Trailer Fields and Transfer Coding

An origin server sends Repr-Digest as trailer field, so it can calculate digest-value while streaming content and thus mitigate resource consumption. The Repr-Digest field-value is the same as in Appendix B.1 because Repr-Digest is designed to be independent of the use of one or more transfer codings (see Section 3).

In the response content below, the string "\r\n" represent the bytes CRLF.

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

Figure 29: GET Request

```
NOTE: '\' line wrapping per RFC 8792

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

8\r\n
{"hello"\r\n
8\r\n
: "world\r\n
3\r\n
"}\n\r\n
0\r\n
Repr-Digest: \
  sha-256=:RK/0qy18MlBSVnWgjwz6lZEWjP/lF5HF9bvEF8FabDg==:\r\n
```

Figure 30: Chunked Response with Digest

Appendix C. Examples of Want-Repr-Digest Solicited Digest

The following examples demonstrate interactions where a client solicits a Repr-Digest using Want-Repr-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 spaces 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-Repr-Digest: sha-256=3, sha=10
```

Figure 31: GET Request with Want-Repr-Digest

```
NOTE: '\' line wrapping per RFC 8792

HTTP/1.1 200 OK
Content-Type: application/json
Repr-Digest: \
  sha-256=:RK/0qy18MlBSVnWgjwz6lZEWjP/lF5HF9bvEF8FabDg==:

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

Figure 32: Response with Different Algorithm

C.2. Server Selects Algorithm Unsupported by Client

The client requests a "sha" digest because that is the only algorithm it supports. The server is not obliged to produce a response containing a "sha" digest, it instead uses a different algorithm.

```
GET /items/123 HTTP/1.1
Host: foo.example
Want-Repr-Digest: sha=10
```

Figure 33: GET Request with Want-Repr-Digest

```
NOTE: '\' line wrapping per RFC 8792

HTTP/1.1 200 OK
Content-Type: application/json
Repr-Digest: \
  sha-512=:YMAam51Jz/jOATT6/zvHrLVgOYTGFy1d6GJiOHTohq4yP+pgk4vf2aCs\
  yRZOtw8MjkM7iw7yZ/WkppmM44T3qg==:

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

Figure 34: Response with Unsupported Algorithm

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

Appendix C.2 is an example where a server ignores the client's preferred digest algorithm. Alternatively a server can also reject the request and return a response with error status code such as 4xx or 5xx. This specification does not prescribe any requirement on status code selection; the follow example illustrates one possible option.

In this example, the client requests a "sha" Repr-Digest, and the server returns an error with problem details [RFC7807] contained in the content. The problem details contain a list of the hashing algorithms that the server supports. This is purely an example, this specification does not define any format or requirements for such content.

```
GET /items/123 HTTP/1.1
Host: foo.example
Want-Repr-Digest: sha=10
```

Figure 35: GET Request with Want-Repr-Digest

```
HTTP/1.1 400 Bad Request
Content-Type: application/problem+json

{
  "title": "Bad Request",
  "detail": "Supported hashing algorithms: sha-256, sha-512",
  "status": 400
}
```

Figure 36: Response advertising the supported algorithms

Appendix D. Sample Digest Values

This section shows examples of digest values for different hashing algorithms. The input value is the JSON object {"hello": "world"}. The digest values are each produced by running the relevant hashing algorithm over the input and running the output bytes through Byte Sequence serialization; see Section 4.1.8 of [STRUCTURED-FIELDS].

```
NOTE: '\' line wrapping per RFC 8792

sha-512 -   :WZDPaVn/7XgHaAy8pmojAkGWoRx2UFChF41A2svX+TaPm+\
            AbwAgBWnrIiYllu7BNNyealdVLvRwEmTHWXvJwew==:

sha-256 -   :X48E9qOokqqrvdts8nOJRJN3OWDUoyWxBf7kbu9DBPE=:

md5 -       :Sd/dVLAcvNLSq16eXua5uQ==:

sha -       :07CavjDP4u3/TungoUHJO/Wzr4c=:

unixsum -   :GQU=:

unixcksum - :7zsHAA==:

adler -     :OZkGFw==:

crc32c -    :Q3lHIA==:
```

Appendix E. Migrating from RFC 3230

HTTP digests are computed by applying a hashing algorithm to input data. RFC 3230 defined the input data as an "instance", a term it also defined. The concept of instance has since been superseded by the HTTP semantic term "representation". It is understood that some implementations of RFC 3230 mistook "instance" to mean HTTP content. Using content for the Digest field is an error that leads to interoperability problems between peers that implement RFC 3230.

RFC 3230 was only ever intended to use what HTTP now defines as selected representation data. The semantic concept of digest and representation are explained alongside the definition of the Repr-Digest field (Section 3).

While the syntax of Digest and Repr-Digest are different, the considerations and examples this document gives for Repr-Digest apply equally to Digest because they operate on the same input data; see Sections 3.1, 6 and 6.3.

RFC 3230 could never communicate the digest of HTTP message content in the Digest field; Content-Digest now provides that capability.

RFC 3230 allowed algorithms to define their output encoding format for use with the Digest field. This resulted in a mix of formats such as base64, hex or decimal. By virtue of using Structured fields, Content-Digest and Repr-Digest use only a single encoding format. Further explanation and examples are provided in Appendix D.

Acknowledgements

This document is based on ideas from [RFC3230], so thanks to Jeff Mogul and Arthur Van Hoff for their great work. The original idea of refreshing RFC3230 arose from an interesting discussion with Mark Nottingham, Jeffrey Yasskin, and Martin 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, Justin Richer, and Erik Wilde.

Code Samples

This section is to be removed before publishing as an RFC.

How can I generate and validate the digest values, computed over the JSON object {"hello": "world"} followed by an LF, 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 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(bytes_, encoding=lambda x: x, algorithm=hashlib.sha256):
    content_encoded = encoding(bytes_)
    return digest_bytes(content_encoded, algorithm)


bytes_ = b'{"hello": "world"}\n'

print("Encoding | hashing algorithm | digest-value")
print("Identity | sha256 |", digest(bytes_))
# Encoding | hashing algorithm | digest-value
# Identity | sha256 | RK/0qy18MlBSVnWgjwz6lZEWjP/lF5HF9bvEF8FabDg=

print("Encoding | hashing algorithm | digest-value")
print("Brotli | sha256 |", digest(bytes_, encoding=brotli.compress))
# Encoding | hashing algorithm | digest-value
# Brotli | sha256 | d435Qo+nKZ+gLcUHn7GQtQ72hiBVAgqoLsZnZPiTGPk=

print("Encoding | hashing algorithm | digest-value")
print("Identity | sha512 |", digest(bytes_, algorithm=hashlib.sha512))
print("Brotli | sha512 |", digest(bytes_, algorithm=hashlib.sha512,
                                    encoding=brotli.compress))
# Encoding | hashing algorithm | digest-value
# Identity | sha512 |b'YMAam51Jz/jOATT6/zvHrLVgOYTGFy1d6GJiOHTohq4yP'
#                      '+pgk4vf2aCsyRZOtw8MjkM7iw7yZ/WkppmM44T3qg=='

# Brotli | sha512 | b'db7fdBbgZMgX1Wb2MjA8zZj+rSNgfmDCEEXM8qLWfpfoNY'
#                     '0sCpHAzZbj09X1/7HAb7Od5Qfto4QpuBsFbUO3dQ=='
```

Changes

This section is to be removed before publishing as an RFC.

Since draft-ietf-httpbis-digest-headers-12

* Be clearer that applications can enforce additional requirements wrt digest
* Change algorithm status names: s/standard/active, s/insecure/deprecated
* Remove "reserved" algorithm status
* Provide clear guidance about the use of standard or deprecated algorithms
* Editorial or minor changes

Since draft-ietf-httpbis-digest-headers-11

* Editorial or minor changes

Since draft-ietf-httpbis-digest-headers-10

* Editorial or minor changes

Since draft-ietf-httpbis-digest-headers-09

* Editorial or minor changes

Since draft-ietf-httpbis-digest-headers-08

* Add note about migrating from RFC 3230. #1968, #1971
* Clarify what Want-* means in responses. #2097
* Editorial changes to structure and to align to HTTP style guide.

Since draft-ietf-httpbis-digest-headers-07

* Introduced Repr-Digest and Want-Repr-Digest, and deprecated Digest and Want-Digest. Use of Structured Fields. #1993, #1919
* IANA refactoring. #1983
* No normative text in security considerations. #1972

Since draft-ietf-httpbis-digest-headers-06

* Remove id-sha-256 and id-sha-512 from the list of supported algorithms #855

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 of 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