Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: November 3, 2021

M. Sporny
Digital Bazaar
L. Rosenthol
Adobe Systems

May 2, 2021

Cryptographic Hyperlinks
draft-sporny-hashlink-07

Abstract

When using a hyperlink to fetch a resource from the Internet, it is often useful to know if the resource has changed since the data was published. Cryptographic hashes, such as SHA-256, are often used to determine if published data has changed in unexpected ways. Due to the nature of most hyperlinks, the cryptographic hash is often published separately from the link itself. This specification describes a data model and serialization formats for expressing cryptographically protected hyperlinks. The mechanisms described in the document enables a system to publish a hyperlink in a way that empowers a consuming application to determine if the resource associated with the hyperlink has changed in unexpected ways.

Feedback

This specification is a work product of the W3C Digital Verification Community Group [1] and the W3C Credentials Community Group [2]. Feedback related to this specification should be logged in the issue tracker [3] or be sent to public-credentials@w3.org [4].

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 November 3, 2021.

Table of Contents

1. Introduction

Uniform Resource Locators (URLs) enable software developers to build distributed systems that are able to publish information using hyperlinks. When a client fetches a resource at the given hyperlink,

the result is typically a stream of data that the client may further process. Due to the design of most hyperlinks, the data associated with a hyperlink may change over time. This design feature is often not an issue for systems that do not depend on static data.

Some software systems expect data published at a specific URL to not change. For example, firmware files, operating system releases, security upgrades, and other high-risk files are often distributed with associated manifest files. These manifest files typically utilize a cryptographic hash per URL to ensure that an attack to modify the files themselves will be detected:

```
b1a653e5...de5d3e8f3  https://example.com/operating-system.iso
7b23bf52...557a0902c  https://example.com/firmware-v4.35.bin
```

An unfortunate downside of the manifest file approach is that a separate system from the URL itself must be utilized to add this level of content integrity protection. In addition, the cryptographic hash format for the files are often application specific and are not easily upgradeable once newer and more advanced cryptographic hash formats are standardized.

New types of distributed file storage networks have been deployed over the past several decades. Examples include HTTP file mirrors for the Debian Operating System, peer-to-peer file networks such as BitTorrent, and content-addressed networks, such as the Inter Planetary File System (IPFS). While each one of these systems have their own URL format, it is currently not possible to express a content-addressed URL that associates the content address to a file published on each one of these networks.

This specification provides a simple data model and serialization formats for cryptographic hyperlinks that:

- Enable existing URLs to add content integrity protection.
- Provide a URL format for multi-sourced content integrity protected data.
- Enable URL metadata to be discarded without having to re-encode the URL.
- Enable algorithm agility for all data model components

1.1. Multiple Encodings

A hashlink can be encoded in two different ways, the RECOMMENDED way to express a hashlink is:

hl:<resource-hash>:<optional-metadata>

To enable existing applications utilizing historical URL schemes to provide content integrity protection, hashlinks may also be encoded using URL parameters:

<url>?hl=<resource-hash>

Implementers should take note that the URL parameter-based encoding mechanism is application specific and SHOULD NOT be used unless the URL resolver for the application cannot be upgraded to support the RECOMMENDED encoding.

2. Hashlink Data Model

The hashlink data model is a simple expression of a cryptographic hash of the resource, one or more URLs, and a content type.

2.1. The Resource Hash

The resource hash is the the mechanism that enables content integrity protection for the associated data stream. The resource hash value MUST be provided in a hashlink.

2.2. The Optional Metadata

All metadata associated with the hashlink is optional and is provided to enable a client to more easily discover data that matches the provided resource hash.

2.2.1. URLs

A hashlink may be associated with a set of one or more URLs that, when dereferenced, result in data that matches the resource hash.

2.2.2. Content Type

A hashlink may be associated with exactly one Content Type that may be used in protocols that support content types, such as HTTP's Accept header.

2.2.3. Experimental Metadata

Application developers often need to express other important metadata related to their specific application. These developers MUST use this field to do so. Data expressed in this field MAY conflict with keys chosen by other developers in other applications. Experimental fields that become widely used are expected to be standardized and become core metadata fields.

3. Hashlink Serialization

A hashlink may be serialized in one or two ways. The first is the RECOMMENDED method, called a "Hashlink URL", which is a compact URL representation of the Hashlink data model. The second is called a "Hashlink as a Parameterized URL", which MUST NOT be used unless there is no mechanism available to upgrade the application's URL resolver.

3.1. Hashlink URL

The beginning of a Hashlink URL always starts with the following three characters:

hl:

The remainder of the URL is a concatenation of the resource hash and, optionally, the Hashlink URL metadata.

3.1.1. Serializing the Resource Hash

The value of the resource hash can be generated by utilizing the following algorithm:

1. Generate the raw hash value by processing the resource data using the cryptographic hashing algorithm.

2. Generate the multihash value by encoding the raw hash using the Multihash Data Format [multihash].

3. Generate the multibase hash by encoding the multihash value using the Multibase Data Format [multibase].

4. Output the multibase hash as the resource hash.

The example below demonstrates the output of the algorithm above for a hashlink that expresses the data "Hello World!" processed using the SHA-2, 256 bit, 32 byte cryptographic algorithm which is then expressed using the base-58 Bitcoin base-encoding format:

zQmWvQxTqbG2Z9HPJgG57jjwR154cKhbtJenbyYTWkjgF3e

3.1.2. Serializing the Metadata

To generate the value for the metadata, the metadata values are encoded in the CBOR Data Format [RFC7049] using the following algorithm:

1. Create the raw output map (CBOR major type 5).

2. If at least one URL exists, add a CBOR key of 15 (0x0f) to the raw output map with a value that is an array (CBOR major type 4).

    1. Encode each URL as a CBOR URI (CBOR type 32) and place it into the array.

3. If the content type exists, add a CBOR key of 14 (0x0e) to the raw output map with a value that is a UTF-8 byte string (0x6) and the value of the content type.

4. If experimental metadata exists, add a CBOR key of 13 (0x0d) and encode it as a map by creating a raw output map (CBOR major type 5). For each item in the map, serialize to CBOR where the CBOR major types, the key name, and the value is derived from the input data. For example a key of "foo" and a value of 200 would be encoded as a CBOR major type of 2 for the key and a CBOR major type of 0 for the value.

5. Generate the multibase value by encoding the raw output map using the Multibase Data Format.

The example below demonstrates the output of the algorithm above for metadata containing a single URL ("http://example.org/hw.txt") with a content type of "text/plain" expressed using the base-58 Bitcoin base-encoding format:

zuh8iaLobXC8g9tfma1CSTtYBakXeSTkHrYA5hmD4F7dCLw8XYwZ1GWyJ3zwF

3.1.3. Deserializing the Metadata

To deserialize the metadata, the "Serializing the Metadata" algorithm is reversed. Implementers MUST use the following table to deserialize keys to JSON:

Key (hex)	JSON key	JSON value
0x0f	"url"	Array of strings
0x0e	"content-type"	string
0x0d	"experimental"	JSON Object

Table 1: Multihash Algorithms Registry

The example below demonstrates the output of the algorithm above for metadata containing a single URL ("http://example.org/hw.txt") with a content type of "text/plain", and an experimental metadata key of "foo" and value of 123:

```
{
  "url": ["http://example.org/hw.txt"],
  "content-type": "text/plain",
  "experimental": {
    "foo": 123
  }
}
```

3.1.4. A Simple Hashlink Example

The example below demonstrates a simple hashlink that provides content integrity protection for the "http://example.org/hw.txt" file, which has a content type of "text/plain" (line breaks added for readability purposes):

```
hl:
zQmWvQxTqbG2Z9HPJgG57jjwR154cKhbtJenbyYTWkjgF3e:
zuh8iaLobXC8g9tfma1CSTtYBakXeSTkHrYA5hmD4F7dCLw8XYwZ1GWyJ3zwF
```

3.2. Hashlink as a Parameterized URL

An algorithm resulting in the same output as the one below MUST be used when encoding the hashlink data model as a set of parameters in a URL:

1. Create an empty string and assign it to the output value.

2. Append the first URL in the URL metadata array to the output URL.

3. Append a URL parameter with a key of "hl" and the value of the resource hash as generated in Section 3.1.1.

3.2.1. Hashlink as a Parameterized URL Example

The example below demonstrates a simple hashlink that provides content integrity protection for the "http://example.org/hw.txt" file, which has a content type of "text/plain":

http://example.org/hw.txt?hl=
zQmWvQxTqbG2Z9HPJgG57jjwR154cKhbtJenbyYTWkjgF3e

4. Hashlink Encoders and Decoders

Hashlink encoders and decoders MUST support the following core algorithms:

1. The SHA-2, 256 bit, 32 byte output cryptographic hashing algorithm and the associated Multihash Data Format.

2. The Bitcoin base58-encoding and decoding algorithm and the associated Multibase Data Format.

Implementations MAY support algorithms and data formats in addition to the ones listed above.

5. Security Considerations

This section documents the security attacks that are out of scope for this specification as well as known attacks and mitigations against those attacks.

5.1. Insecure Hashing Functions

There are a number of insecure cryptographic hashing functions in deployment today. Among these are MD5 and SHA-1. Implementers MUST throw an error by default when encoding or decoding these values. Implementers MAY provide a non-default library option to override the error.

6. References

6.1. Normative References

[multibase]
Benet, J. and M. Sporny, "The Multihash Data Format", December 2018, <https://tools.ietf.org/id/draft-multiformats-multibase-00.html>.

[multihash] Benet, J. and M. Sporny, "The Multihash Data Format", August 2018, <https://tools.ietf.org/id/draft-multiformats-multihash-00.html>.

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

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

6.2. URIs

[1] https://w3c-dvcg.github.io/

[2] https://w3c-ccg.github.io/

[3] https://github.com/w3c-dvcg/hashlink/issues

[4] mailto:public-credentials@w3.org

Appendix A. Security Considerations

There are a number of security considerations to take into account when implementing or utilizing this specification: TBD

Appendix B. Test Values

The following test values may be used to verify the conformance of Hashlink encoders and decoders.

B.1. Simple Hashlink URL

The following Hashlink URL encodes the data "Hello World!" served from the "http://example.org/hw.txt" URL with a content type of "text/plain". The resource hash is generated using the SHA-2, 256 bit, 32 byte cryptographic algorithm which is then encoded using the base-58 Bitcoin base-encoding format. The metadata options are encoded using the base-58 Bitcoin base-encoding format. The final Hashlink URL is (new lines added for readability purposes):

```
hl:
zQmWvQxTqbG2Z9HPJgG57jjwR154cKhbtJenbyYTWkjgF3e:
zuh8iaLobXC8g9tfma1CSTtYBakXeSTkHrYA5hmD4F7dCLw8XYwZ1GWyJ3zwF
```

B.2. Multi-sourced Hashlink URL

The following Hashlink URL encodes the data "Hello World!" served from three different networks. The first is a standard Web-based URL ("http://example.org/hw.txt"), the second is an IPFS-based URL ("ipfs:/ipfs/QmXfrS3pHerg44zzK6QKQj6JDk8H6cMtQS7pdXbohwNQfK/hello"), and the third is a Tor-based URL ("http://c4m3g2upq6pkufl4.onion/hworld.txt"). The resource hash is generated using the SHA-2, 256 bit, 32 byte cryptographic algorithm which is then encoded using the base-58 Bitcoin base-encoding format. The metadata options are encoded using the base-58 Bitcoin base-encoding format. The final Hashlink URL is (new lines added for readability purposes):

hl:
zQmWvQxTqbG2Z9HPJgG57jjwR154cKhbtJenbyYTWkjgF3e:
z333PdTakFeJueF2bim3PaaDqbtqjkpxUc8ETSWXe6dQLWXQWvqiUdw8TJrncx3uKhwfc
88MtM5xZbR27FhVRUKv9ogekamVtdE3UbXnXpMRT1AseCtoBUt1NE8x2SsnJxGfiZN45V
VSCp6jh4dgcufL16tWrHREiSYESEGP1J75yXCvAdvKPr7nb5aYujLeay8Ww

Appendix C. Acknowledgements

The editors would like to thank the following individuals for feedback on and implementations of the specification (in alphabetical order): TBD

Portions of the work on this specification have been funded by the United States Department of Homeland Security's Science and Technology Directorate under contract HSHQDC-17-C-00019. The content of this specification does not necessarily reflect the position or the policy of the U.S. Government and no official endorsement should be inferred.

Authors' Addresses

Manu Sporny
Digital Bazaar
203 Roanoke Street W.
Blacksburg, VA 24060
US

Phone: +1 540 961 4469
Email: msporny@digitalbazaar.com
URI: http://manu.sporny.org/

Leonard Rosenthol
Adobe Systems
345 Park Ave.
San Jose, CA 95110-2704
US

Phone: +1 800 833 6687
Email: lrosenth@adobe.com
URI: https://www.linkedin.com/in/lrosenthol/