Network Working Group M. Sporny
Internet-Draft Digital Bazaar
Intended status: Standards Track December 30, 2018
Expires: July 3, 2019

Cryptographic Hyperlinks
draft-sporny-hashlink-00

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

Copyright Notice

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:

o Enable existing URLs to add content integrity protection.

o Provide a URL format for multi-sourced content integrity protected data.

o Enable URL metadata to be discarded without having to re-encode the URL.

o 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.

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:

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:

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 using the following algorithm:

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. 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:

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:

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

5. References

5.1. 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>.

5.2. Informative References

5.3. URIs

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

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

[3] https://github.com/w3c-dvcg/multibase/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):

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):

Appendix C. Acknowledgements

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

Author's Address

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/