ï»¿

Internet Engineering Task Force (IETF)
Request for Comments: 9054
Category: Informational
ISSN: 2070-1721

J. Schaad
August Cellars

August 2022

CBOR Object Signing and Encryption (COSE): Hash Algorithms

Abstract

The CBOR Object Signing and Encryption (COSE) syntax (see RFC 9052) does not define any direct methods for using hash algorithms. There are, however, circumstances where hash algorithms are used, such as indirect signatures, where the hash of one or more contents are signed, and identification of an X.509 certificate or other object by the use of a fingerprint. This document defines hash algorithms that are identified by COSE algorithm identifiers.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

The CBOR Object Signing and Encryption (COSE) syntax [RFC9052] does not define any direct methods for the use of hash algorithms. It also does not define a structure syntax that is used to encode a digested object structure along the lines of the DigestedData ASN.1 structure in [CMS]. This omission was intentional, as a structure consisting of just a digest identifier, the content, and a digest value does not, by itself, provide any strong security service. Additionally, an application is going to be better off defining this type of structure so that it can include any additional data that needs to be hashed, as well as methods of obtaining the data.

While the above is true, there are some cases where having some standard hash algorithms defined for COSE with a common identifier makes a great deal of sense. Two of the cases where these are going to be used are:

* Indirect signing of content, and

* Object identification.

Indirect signing of content is a paradigm where the content is not directly signed, but instead a hash of the content is computed, and that hash value -- along with an identifier for the hash algorithm -- is included in the content that will be signed. Indirect signing allows for a signature to be validated without first downloading all of the content associated with the signature. Rather, the signature can be validated on all of the hash values and pointers to the associated contents; those associated parts can then be downloaded, then the hash value of that part can be computed and compared to the hash value in the signed content. This capability can be of even greater importance in a constrained environment, as not all of the content signed may be needed by the device. An example of how this is used can be found in Section 5.4 of [SUIT-MANIFEST].

The use of hashes to identify objects is something that has been very common. One of the primary things that has been identified by a hash function in a secure message is a certificate. Two examples of this can be found in [ESS] and the COSE equivalents in [COSE-x509].

1.1. Requirements Terminology

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.

2. Hash Algorithm Usage

As noted in the previous section, hash functions can be used for a variety of purposes. Some of these purposes require that a hash function be cryptographically strong. These include direct and indirect signatures -- that is, using the hash as part of the signature or using the hash as part of the body to be signed. Other uses of hash functions may not require the same level of strength.

This document contains some hash functions that are not designed to be used for cryptographic operations. An application that is using a hash function needs to carefully evaluate exactly what hash properties are needed and which hash functions are going to provide them. Applications should also make sure that the ability to change hash functions is part of the base design, as cryptographic advances are sure to reduce the strength of any given hash function [BCP201].

A hash function is a map from one, normally large, bit string to a second, usually smaller, bit string. As the number of possible input values is far greater than the number of possible output values, it is inevitable that there are going to be collisions. The trick is to make sure that it is difficult to find two values that are going to map to the same output value. A "Collision Attack" is one where an attacker can find two different messages that have the same hash value. A hash function that is susceptible to practical collision attacks SHOULD NOT be used for a cryptographic purpose. The discovery of theoretical collision attacks against a given hash function SHOULD trigger protocol maintainers and users to review the continued suitability of the algorithm if alternatives are available and migration is viable. The only reason such a hash function is used is when there is absolutely no other choice (e.g., a Hardware Security Module (HSM) that cannot be replaced), and only after looking at the possible security issues. Cryptographic purposes would include the creation of signatures or the use of hashes for indirect signatures. These functions may still be usable for noncryptographic purposes.

An example of a noncryptographic use of a hash is filtering from a collection of values to find a set of possible candidates; the candidates can then be checked to see if they can successfully be used. A simple example of this is the classic fingerprint of a certificate. If the fingerprint is used to verify that it is the correct certificate, then that usage is a cryptographic one and is subject to the warning above about collision attack. If, however, the fingerprint is used to sort through a collection of certificates to find those that might be used for the purpose of verifying a signature, a simple filter capability is sufficient. In this case, one still needs to confirm that the public key validates the signature (and that the certificate is trusted), and all certificates that don't contain a key that validates the signature can be discarded as false positives.

To distinguish between these two cases, a new value in the Recommended column of the "COSE Algorithms" registry has been added. "Filter Only" indicates that the only purpose of a hash function should be to filter results; it is not intended for applications that require a cryptographically strong algorithm.

2.1. Example CBOR Hash Structure

[COSE] did not provide a default structure for holding a hash value both because no separate hash algorithms were defined and because the way the structure is set up is frequently application specific. There are four fields that are often included as part of a hash structure:

* The hash algorithm identifier.

* The hash value.

* A pointer to the value that was hashed. This could be a pointer to a file, an object that can be obtained from the network, a pointer to someplace in the message, or something very application specific.

* Additional data. This can be something as simple as a random value (i.e., salt) to make finding hash collisions slightly harder (because the payload handed to the application could have been selected to have a collision), or as complicated as a set of processing instructions that is used with the object that is pointed to. The additional data can be dealt with in a number of ways, prepending or appending to the content, but it is strongly suggested that either it be a fixed known size, or the lengths of the pieces being hashed be included so that the resulting byte string has a unique interpretation as the additional data. (Encoding as a CBOR array accomplishes this requirement.)

An example of a structure that permits all of the above fields to exist would look like the following:

```
COSE_Hash_V = (
    1 : int / tstr, # Algorithm identifier
    2 : bstr, # Hash value
```

```
    ? 3 : tstr, # Location of object that was hashed
    ? 4 : any   # object containing other details and things
    )
```

Below is an alternative structure that could be used in situations where one is searching a group of objects for a matching hash value. In this case, the location would not be needed, and adding extra data to the hash would be counterproductive. This results in a structure that looks like this:

```
COSE_Hash_Find = [
    hashAlg : int / tstr,
    hashValue : bstr
]
```

3. Hash Algorithm Identifiers

3.1. SHA-1 Hash Algorithm

The SHA-1 hash algorithm [RFC3174] was designed by the United States National Security Agency and published in 1995. Since that time, a large amount of cryptographic analysis has been applied to this algorithm, and a successful collision attack has been created [SHA-1-collision]. The IETF formally started discouraging the use of SHA-1 in [RFC6194].

Despite these facts, there are still times where SHA-1 needs to be used; therefore, it makes sense to assign a code point for the use of this hash algorithm. Some of these situations involve historic HSMs where only SHA-1 is implemented; in other situations, the SHA-1 value is used for the purpose of filtering; thus, the collision-resistance property is not needed.

Because of the known issues for SHA-1 and the fact that it should no longer be used, the algorithm will be registered with the recommendation of "Filter Only". This provides guidance about when the algorithm is safe for use, while discouraging usage where it is not safe.

The COSE capabilities for this algorithm is an empty array.

Name	Value	Description	Capabilities	Reference	Recommended
SHA-1	-14	SHA-1 Hash	[]	RFC 9054	Filter Only

Table 1: SHA-1 Hash Algorithm

3.2. SHA-2 Hash Algorithms

The family of SHA-2 hash algorithms [FIPS-180-4] was designed by the United States National Security Agency and published in 2001. Since that time, some additional algorithms have been added to the original set to deal with length-extension attacks and some performance issues. While the SHA-3 hash algorithms have been published since that time, the SHA-2 algorithms are still broadly used.

There are a number of different parameters for the SHA-2 hash functions. The set of hash functions that has been chosen for inclusion in this document is based on those different parameters and some of the trade-offs involved.

* *SHA-256/64* provides a truncated hash. The length of the truncation is designed to allow for smaller transmission size. The trade-off is that the odds that a collision will occur increase proportionally. Use of this hash function requires analysis of the potential problems that could result from a collision, or it must be limited to where the purpose of the hash is noncryptographic.

The latter is the case for some of the scenarios identified in [COSE-x509], specifically, for the cases when the hash value is used to select among possible certificates: if there are multiple choices remaining, then each choice can be tested by using the public key.

* *SHA-256* is probably the most common hash function used currently. SHA-256 is an efficient hash algorithm for 32-bit hardware.

* *SHA-384* and *SHA-512* hash functions are efficient for 64-bit hardware.

* *SHA-512/256* provides a hash function that runs more efficiently on 64-bit hardware but offers the same security level as SHA-256.

> NOTE: SHA-256/64 is a simple truncation of SHA-256 to 64 bits defined in this specification. SHA-512/256 is a modified variant of SHA-512 truncated to 256 bits, as defined in [FIPS-180-4].

The COSE capabilities array for these algorithms is empty.

Name	Value	Description	Capabilities	Reference	Recommended
SHA-256/64	-15	SHA-2 256-bit Hash truncated to 64-bits	[]	RFC 9054	Filter Only
SHA-256	-16	SHA-2 256-bit Hash	[]	RFC 9054	Yes
SHA-384	-43	SHA-2 384-bit Hash	[]	RFC 9054	Yes
SHA-512	-44	SHA-2 512-bit Hash	[]	RFC 9054	Yes
SHA-512/256	-17	SHA-2 512-bit Hash truncated to 256-bits	[]	RFC 9054	Yes

Table 2: SHA-2 Hash Algorithms

3.3. SHAKE Algorithms

The family of SHA-3 hash algorithms [FIPS-202] was the result of a competition run by NIST. The pair of algorithms known as SHAKE-128 and SHAKE-256 are the instances of SHA-3 that are currently being standardized in the IETF. This is the reason for including these algorithms in this document.

The SHA-3 hash algorithms have a significantly different structure than the SHA-2 hash algorithms.

Unlike the SHA-2 hash functions, no algorithm identifier is created for shorter lengths. The length of the hash value stored is 256 bits for SHAKE-128 and 512 bits for SHAKE-256.

The COSE capabilities array for these algorithms is empty.

Name	Value	Description	Capabilities	Reference	Recommended
SHAKE128	-18	SHAKE-128 256-bit Hash Value	[]	RFC 9054	Yes
SHAKE256	-45	SHAKE-256 512-bit Hash Value	[]	RFC 9054	Yes

Table 3: SHAKE Hash Functions

4. IANA Considerations

4.1. COSE Algorithm Registry

IANA has registered the following algorithms in the "COSE Algorithms" registry (https://www.iana.org/assignments/cose/).

* The SHA-1 hash function found in Table 1.

* The set of SHA-2 hash functions found in Table 2.

* The set of SHAKE hash functions found in Table 3.

Many of the hash values produced are relatively long; as such, use of a two-byte algorithm identifier seems reasonable. SHA-1 is tagged as "Filter Only", so a longer algorithm identifier is appropriate even though it is a shorter hash value.

IANA has added the value of "Filter Only" to the set of legal values for the Recommended column. This value is only to be used for hash functions and indicates that it is not to be used for purposes that require collision resistance. As a result of this addition, IANA has added this document as a reference for the "COSE Algorithms" registry.

5. Security Considerations

Protocols need to perform a careful analysis of the properties of a hash function that are needed and how they map onto the possible attacks. In particular, one needs to distinguish between those uses that need the cryptographic properties, such as collision resistance, and uses that only need properties that correspond to possible object identification. The different attacks correspond to who or what is being protected: is it the originator that is the attacker or a third party? This is the difference between collision resistance and second pre-image resistance. As a general rule, longer hash values are "better" than short ones, but trade-offs of transmission size, timeliness, and security all need to be included as part of this analysis. In many cases, the value being hashed is a public value and, as such, (first) pre-image resistance is not part of this analysis.

Algorithm agility needs to be considered a requirement for any use of hash functions [BCP201]. As with any cryptographic function, hash functions are under constant attack, and the cryptographic strength of hash algorithms will be reduced over time.

6. References

6.1. Normative References

[FIPS-180-4]
NIST, "Secure Hash Standard", FIPS PUB 180-4, DOI 10.6028/NIST.FIPS.180-4, August 2015, <https://doi.org/10.6028/NIST.FIPS.180-4>.

[FIPS-202] Dworkin, M.J., "SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions", FIPS PUB 202, DOI 10.6028/NIST.FIPS.202, August 2015, <https://doi.org/10.6028/NIST.FIPS.202>.

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

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

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC9052] Schaad, J., "CBOR Object Signing and Encryption (COSE): Structures and Process", STD 96, RFC 9052, DOI 10.17487/RFC9052, August 2022, <https://www.rfc-editor.org/info/rfc9052>.

6.2. Informative References

[BCP201] Housley, R., "Guidelines for Cryptographic Algorithm Agility and Selecting Mandatory-to-Implement Algorithms", BCP 201, RFC 7696, November 2015, <https://www.rfc-editor.org/info/bcp201>.

[CMS] Housley, R., "Cryptographic Message Syntax (CMS)", STD 70, RFC 5652, DOI 10.17487/RFC5652, September 2009, <https://www.rfc-editor.org/info/rfc5652>.

[COSE] Schaad, J., "CBOR Object Signing and Encryption (COSE)", RFC 8152, DOI 10.17487/RFC8152, July 2017, <https://www.rfc-editor.org/info/rfc8152>.

[COSE-x509]
Schaad, J., "CBOR Object Signing and Encryption (COSE): Header parameters for carrying and referencing X.509 certificates", Work in Progress, Internet-Draft, draft-ietf-cose-x509-08, 14 December 2020, <https://datatracker.ietf.org/doc/html/draft-ietf-cose-x509-08>.

[ESS] Hoffman, P., Ed., "Enhanced Security Services for S/MIME", RFC 2634, DOI 10.17487/RFC2634, June 1999, <https://www.rfc-editor.org/info/rfc2634>.

[RFC6194] 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/info/rfc6194>.

[SHA-1-collision]
Stevens, M., Bursztein, E., Karpman, P., Albertini, A., and Y. Markov, "The first collision for full SHA-1", February 2017, <https://shattered.io/static/shattered.pdf>.

[SUIT-MANIFEST]
Moran, B., Tschofenig, H., Birkholz, H., and K. Zandberg, "A Concise Binary Object Representation (CBOR)-based Serialization Format for the Software Updates for Internet of Things (SUIT) Manifest", Work in Progress, Internet-Draft, draft-ietf-suit-manifest-19, 9 August 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-suit-manifest-19>.

Author's Address

Jim Schaad