CBOR Object Signing and Encryption (COSE): Hash Algorithms

Abstract

The CBOR Object Signing and Encryption (COSE) syntax [I-D.ietf-cose-rfc8152bis-struct] does not define any direct methods for using hash algorithms. There are however circumstances where hash algorithms are used: Indirect signatures where the hash of one or more contents are signed. X.509 certificate or other object identification by the use of a fingerprint. This document defines a set of hash algorithms that are identified by COSE Algorithm Identifiers.

Contributing to this document

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

The source for this draft is being maintained in GitHub. Suggested changes should be submitted as pull requests at https://github.com/cose-wg/X509 Editorial changes can be managed in GitHub, but any substantial issues need to be discussed on the COSE mailing list.

Table of Contents

1. **Introduction**

The CBOR Object Signing and Encryption (COSE) syntax 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 the hash algorithm, is included in the
content that will be signed. Doing indirect signing allows for a
signature to be validated without first downloading all of the
content associated with the signature. 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.

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 for secure message is a certificate. Two examples of
this can be found in [ESS] and the newly defined COSE equivalents in
[I-D.ietf-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 do 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 a hash function.

A hash function is a map from one, normally large, bit string to a
second, usually smaller, bit string. There are going to be
collisions by a hash function, the trick is to make sure that it is
difficult to find two values that are going to map to the same
output value. The standard "Collision Attack" is one where an
attacker can find two different messages that have the same hash
value. If a collision attack exists, then the function SHOULD NOT be
used for a cryptographic purpose. The only reason why 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 non-cryptographic purposes.

An example of a non-cryptographic use of a hash is for filtering
from a collection of values to find possible candidates that can
later be checked to see if they are the correct one. 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 subject to a collision attack as above. 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 validate that the public key validates
the signature (and 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 is to be added.
"Filter Only" indicates that the only purpose of a hash function
should be to filter results and not those which require collision
resistance.

2.1. Example CBOR hash structure

[COSE] did not provide a default structure for holding a hash value
not only because no separate hash algorithms were defined, but
because how the structure is setup 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, or a
   pointer to someplace in the message, or something very
   application specific.

  *Additional data, this can be something as simple as a random
   value to make finding hash collisions slightly harder (as the
   value handed to the application cannot have been selected to have
   a collision), or as complicated as a set of processing
   instructions that are 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 to it either be a fixed known size, or the lengths of
   the pieces being hashed be included. (Encoding as a CBOR array
   accomplished this requirement.)

An example of a structure which 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 hashed
    4 : any ?   # object containing other details and things
    )
```

An alternate structure that could be used for situations where one
is searching a group of objects for a match. 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 with the publishing of [RFC6194].

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

Because of the known issues for SHA-1 and the fact that is should no longer be used, the algorithm will be registered with the recommendation of "Filter Only".

The COSE capabilities for this algorithm is an empty array.

Name	Value	Description	Capabilities	Reference	Recommended
SHA-1	TBD6	SHA-1 Hash	[]	[This Document]	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 which have been chosen for inclusion in this document are 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. Locations that use this hash function need either to analysis the potential problems with having a collision occur, or where the only function of the hash is to narrow the possible choices.

The latter is the case for [I-D.ietf-cose-x509], the hash value
is used to select possible certificates and, if there are
multiple choices 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 levels as
    SHA-256.

The COSE capabilities for these algorithms is an empty array.

Name	Value	Description	Capabilities	Reference	Recommended
SHA-256/64	TBD1	SHA-2 256-bit Hash truncated to 64-bits	[]	[This Document]	Filter Only
SHA-256	TBD2	SHA-2 256-bit Hash	[]	[This Document]	Yes
SHA-384	TBD3	SHA-2 384-bit Hash	[]	[This Document]	Yes
SHA-512	TBD4	SHA-2 512-bit Hash	[]	[This Document]	Yes
SHA-512/256	TBD5	SHA-2 512-bit Hash truncated to 256-bits	[]	[This Document]	Yes

Table 2: SHA-2 Hash Algorithms

3.3. SHAKE Algorithms

The family 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.

The SHA-3 hash algorithms have a significantly different structure
than the SHA-2 hash algorithms. One of the benefits of this
differences is that when computing a shorter SHAKE hash value, the
value is not a prefix of the result of computing the longer hash.

Unlike the SHA-2 hash functions, no algorithm identifier is created
for shorter lengths. Applications can specify a minimum length for

any hash function. A validator can infer the actual length from the
hash value in these cases.

The COSE capabilities for these algorithms is an empty array.

Name	Value	Description	Capabilities	Reference	Recommended
SHAKE128	TBD10	128-bit SHAKE	[]	[This Document]	Yes
SHAKE256	TBD11	256-bit SHAKE	[]	[This Document]	Yes

Table 3: SHAKE Hash Functions

4. IANA Considerations

4.1. COSE Algorithm Registry

IANA is requested to register the following algorithms in the "COSE
Algorithms" registry.

  *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 and as such the
use of a two byte algorithm identifier seems reasonable. SHA-1 is
tagged as deprecated and thus a longer algorithm identifier is
appropriate even though it is a shorter hash value.

In addition, IANA is to add 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 which require collision resistance.

5. Security Considerations

The security considerations have already been called out as part of
the previous text. The following issues need to be dealt with:

  *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, i.e. collision
   resistance, and 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, as such pre-image resistance is not
part of this analysis.

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

6. Normative References

[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>.

[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>.

[I-D.ietf-cose-rfc8152bis-struct]
           Schaad, J., "CBOR Object Signing and Encryption (COSE):
           Structures and Process", Work in Progress, Internet-
           Draft, draft-ietf-cose-rfc8152bis-struct-07, 17 November
           2019, <https://tools.ietf.org/html/draft-ietf-cose-
           rfc8152bis-struct-07>.

[FIPS-180-4] National Institute of Standards and Technology, "Secure
           Hash Standard", FIPS PUB 180-4, August 2015.

[FIPS-202] National Institute of Standards and Technology, "SHA-3
           Standard: Permutation-Based Hash and Extendable-Output
           Functions", FIPS PUB 202, August 2015.

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

7. Informative References

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

[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>.

[I-D.ietf-cose-x509]
              Schaad, J., "CBOR Object Signing and Encryption
              (COSE): Headers for carrying and referencing X.509
              certificates", Work in Progress, Internet-Draft, draft-
              ietf-cose-x509-05, 4 November 2019, <https://
              tools.ietf.org/html/draft-ietf-cose-x509-05>.

[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>.

[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>.

Author's Address

Jim Schaad
August Cellars

Email: ietf@augustcellars.com