CBOR Object Signing and Encryption (COSE): Hash Algorithms
draft-ietf-cose-hash-algs-01

Abstract

The CBOR Object Signing and Encryption (COSE) syntax RFC 8152 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 external contents are signed, or thumbprints, for identification of X.509 certificates or other objects. This document defines a set of hash algorithms that are identified by COSE Algorithm Identifiers.

Contributing to this document

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.

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 December 12, 2019.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.
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 add 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 the 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.

1.2. Open Issues

RFC Editor: Please remove this section before publishing.

* No Open Issues

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 strength of a hash function.

A hash function is a map from a large bit string to a 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 HSM that cannot be replaced), and only after looking at the possible security issues. Cryptographic purposes would include 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 thumbprint of a certificate. If the thumbprint is used to verify that it is the correct certificate, then that usage is subject to a collision attack as above. If however, the thumbprint 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 to be used with the object that is pointed to.

An example of a structure which permits all of the above fields to
exist would look like the following. There is no definition here of
what goes into the ‘any’ value and how it would be included in the
computed hash value.

COSE_Hash_V = ( 1 : int / tstr, # Algorithm identifier 2 : bstr, # Hash value
3 : tstr ?, # Location of object hashed 4 : any ?   # object containing other d
tails and things - prefixed to the object to be hashed )

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".
### 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 has 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 analyze 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.
Table 2: SHA-2 Hash Algorithms

3.3. SHAKE Algorithms

The family SHA-3 hash algorithms [FIPS-180-4] 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 difference 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 what the minimum length for a hash function for the protocol. A validator can infer the actual length from the hash value in these cases.
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


7. Informative References


Author’s Address

Jim Schaad
August Cellars

Email: ietf@augustcellars.com
COSE and JOSE Registrations for WebAuthn Algorithms
draft-ietf-cose-webauthn-algorithms-01

Abstract

The W3C Web Authentication (WebAuthn) specification and the FIDO2 Client to Authenticator Protocol (CTAP) specification use COSE algorithm identifiers. This specification registers algorithms in the IANA "COSE Algorithms" registry that are used by WebAuthn and CTAP implementations that are not already registered. Also, they are registered in the IANA "JSON Web Signature and Encryption Algorithms" registry, when not already registered there.

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 January 9, 2020.

Copyright Notice

Copyright (c) 2019 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of
1. Introduction

This specification defines how to use several algorithms with COSE [RFC8152] that are used by implementations of the W3C Web Authentication (WebAuthn) [WebAuthn] and FIDO2 Client to Authenticator Protocol (CTAP) [CTAP] specifications. These algorithms are registered in the IANA "COSE Algorithms" registry [IANA.COSE.Algorithms] and also in the IANA "JSON Web Signature and Encryption Algorithms" registry [IANA.JOSE.Algorithms], when not already registered there.

1.1. Requirements Notation and 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.
2. RSASSA-PKCS1-v1_5 Signature Algorithm

The RSASSA-PKCS1-v1_5 signature algorithm is defined in [RFC8017]. The RSASSA-PKCS1-v1_5 signature algorithm is parameterized with a hash function (h).

A key of size 2048 bits or larger MUST be used with these algorithms. Implementations need to check that the key type is 'RSA' when creating or verifying a signature.

The RSASSA-PKCS1-v1_5 algorithms specified in this document are in the following table.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Hash</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RS256</td>
<td>TBD (temporary assignment</td>
<td>SHA-256</td>
<td>RSASSA-PKCS1-v1_5</td>
</tr>
<tr>
<td></td>
<td>-257 already in place)</td>
<td></td>
<td>using SHA-256</td>
</tr>
<tr>
<td>RS384</td>
<td>TBD (temporary assignment</td>
<td>SHA-384</td>
<td>RSASSA-PKCS1-v1_5</td>
</tr>
<tr>
<td></td>
<td>-288 already in place)</td>
<td></td>
<td>using SHA-384</td>
</tr>
<tr>
<td>RS512</td>
<td>TBD (temporary assignment</td>
<td>SHA-512</td>
<td>RSASSA-PKCS1-v1_5</td>
</tr>
<tr>
<td></td>
<td>-259 already in place)</td>
<td></td>
<td>using SHA-512</td>
</tr>
<tr>
<td>RS1</td>
<td>TBD (temporary assignment</td>
<td>SHA-1</td>
<td>RSASSA-PKCS1-v1_5</td>
</tr>
<tr>
<td></td>
<td>-65535 already in place)</td>
<td></td>
<td>using SHA-1</td>
</tr>
</tbody>
</table>

Table 1: RSASSA-PKCS1-v1_5 Algorithm Values

3. Using secp256k1 with JOSE and COSE

This section defines algorithm encodings and representations enabling the Standards for Efficient Cryptography Group (SECG) elliptic curve secp256k1 [SEC2] to be used for JSON Object Signing and Encryption (JOSE) [RFC7515] and CBOR Object Signing and Encryption (COSE) [RFC8152] messages.

3.1. JOSE and COSE secp256k1 Curve Key Representations

The Standards for Efficient Cryptography Group (SECG) elliptic curve secp256k1 [SEC2] is represented in a JSON Web Key (JWK) [RFC7517] using these values:

- "kty": "EC"
- "crv": "secp256k1"

plus "x" and "y" values to represent the curve point for the key. Other optional values such as "alg" MAY also be present.
It is represented in a COSE_key [RFC8152] using these values:

- "kty" (1): "EC2" (2)
- "crv" (-1): "secp256k1" (TBD - requested assignment 8)

plus "x" (-2) and "y" (-3) values to represent the curve point for the key. Other optional values such as "alg" (3) MAY also be present.

3.2. ECDSA Signature with secp256k1 Curve

The ECDSA signature algorithm is defined in [DSS]. This specification defines the use of ECDSA with the secp256k1 curve and the SHA-256 [DSS] cryptographic hash function. Implementations need to check that the key type is "EC" for JOSE or "EC2" (2) for COSE when creating or verifying a signature.

The ECDSA secp256k1 SHA-256 digital signature is generated as follows:

1. Generate a digital signature of the JWS Signing Input or the COSE payload using ECDSA secp256k1 SHA-256 with the desired private key. The output will be the pair (R, S), where R and S are 256-bit unsigned integers.

2. Turn R and S into octet sequences in big-endian order, with each array being be 32 octets long. The octet sequence representations MUST NOT be shortened to omit any leading zero octets contained in the values.

3. Concatenate the two octet sequences in the order R and then S. (Note that many ECDSA implementations will directly produce this concatenation as their output.)

4. The resulting 64-octet sequence is the JWS Signature or COSE signature value.

The ECDSA secp256k1 SHA-256 algorithm specified in this document uses these identifiers:
4. IANA Considerations

4.1. COSE Algorithms Registrations

This section registers the following values in the IANA "COSE Algorithms" registry [IANA.COSE.Algorithms].

- **Name**: RS256
  - **Value**: TBD (temporary assignment -257 already in place)
  - **Description**: RSASSA-PKCS1-v1_5 using SHA-256
  - **Reference**: Section 2 of this document
  - **Recommended**: No

- **Name**: RS384
  - **Value**: TBD (temporary assignment -258 already in place)
  - **Description**: RSASSA-PKCS1-v1_5 using SHA-384
  - **Reference**: Section 2 of this document
  - **Recommended**: No

- **Name**: RS512
  - **Value**: TBD (temporary assignment -259 already in place)
  - **Description**: RSASSA-PKCS1-v1_5 using SHA-512
  - **Reference**: Section 2 of this document
  - **Recommended**: No

- **Name**: RS1
  - **Value**: TBD (temporary assignment -65535 already in place)
  - **Description**: RSASSA-PKCS1-v1_5 using SHA-1
  - **Reference**: Section 2 of this document
  - **Recommended**: Deprecated

- **Name**: ES256K
  - **Value**: TBD (requested assignment -43)
  - **Description**: ECDSA using secp256k1 curve and SHA-256
  - **Reference**: Section 3.2 of this document
  - **Recommended**: Yes
4.2. COSE Elliptic Curves Registrations

This section registers the following value in the IANA "COSE Elliptic Curves" registry [IANA.COSE.Curves].

- Name: secp256k1
- Value: TBD (requested assignment 8)
- Key Type: EC2
- Description: SECG secp256k1 curve
- Change Controller: IESG
- Reference: Section 3.1 of [[ this specification ]]
- Recommended: Yes

4.3. JOSE Algorithms Registrations

This section registers the following value in the IANA "JSON Web Signature and Encryption Algorithms" registry [IANA.JOSE.Algorithms].

- Algorithm Name: ES256K
- Algorithm Description: ECDSA using secp256k1 curve and SHA-256
- Algorithm Usage Locations: alg
- JOSE Implementation Requirements: Optional
- Change Controller: IESG
- Reference: Section 3.2 of [[ this specification ]]
- Algorithm Analysis Document(s): [SEC2]

4.4. JSON Web Key Elliptic Curves Registrations

This section registers the following value in the IANA "JSON Web Key Elliptic Curve" registry [IANA.JOSE.Curves].

- Curve Name: secp256k1
- Curve Description: SECG secp256k1 curve
- JOSE Implementation Requirements: Optional
- Change Controller: IESG
- Specification Document(s): Section 3.1 of [[ this specification ]]

5. Security Considerations

5.1. RSA Key Size Security Considerations

The security considerations on key sizes for RSA algorithms from Section 6.1 of [RFC8230] also apply to the RSA algorithms in this specification.
5.2. RSASSA-PKCS1-v1_5 with SHA-2 Security Considerations

The security considerations on the use of RSASSA-PKCS1-v1_5 with SHA-2 hash functions from Section 8.3 of [RFC7518] also apply to their use in this specification. For that reason, these algorithms are registered as being "Not Recommended".

5.3. RSASSA-PKCS1-v1_5 with SHA-1 Security Considerations

The security considerations on the use of the SHA-1 hash function from [RFC6194] apply in this specification. For that reason, the "RS1" algorithm is registered as "Deprecated". It MUST NOT be used by COSE implementations.

A COSE algorithm identifier for this algorithm is nonetheless being registered because deployed TPMs continue to use it, and therefore WebAuthn implementations need a COSE algorithm identifier for "RS1" when TPM attestations using this algorithm are being represented.

5.4. secp256k1 Security Considerations

Care should be taken that a secp256k1 key is not mistaken for a P-256 key, given that their representations are the same except for the "crv" value.

The procedures and security considerations described in the [SEC1], [SEC2], and [DSS] specifications apply to implementations of this specification.

6. References

6.1. Normative References


6.2. Informative References

Jones                    Expires January 9, 2020                [Page 8]


[IANA.JOSE.Curves] IANA, "JSON Web Key Elliptic Curve", <https://www.iana.org/assignments/jose/jose.xhtml#web-key-elliptic-curve>.


Acknowledgements

Thanks to Stephen Farrell, John Fontana, Jeff Hodges, John Mattsson, Tony Nadalin, Matt Palmer, Jim Schaad, Goeran Selander, Wendy Seltzer, Sean Turner, and Samuel Weiler for their roles in registering these algorithm identifiers.

Document History

[[ to be removed by the RFC Editor before publication as an RFC ]]

-01
Internet-DraCOSE and JOSE Registrations for WebAuthn Algorithm July 2019

- Changed the JOSE curve identifier from "P-256K" to "secp256k1".
- Specified that secp256k1 signing is done using the SHA-256 hash function.
- Created the initial working group draft from draft-jones-cose-additional-algorithms-00, changing only the title, date, and history entry.

Author’s Address

Michael B. Jones
Microsoft

Email: mbj@microsoft.com
URI: http://self-issued.info/
CBOR Object Signing and Encryption (COSE): Headers for carrying and referencing X.509 certificates

draft-ietf-cose-x509-03

Abstract

The CBOR Signing And Encrypted Message (COSE) structure uses references to keys in general. For some algorithms, additional properties are defined which carry parts of keys as needed. The COSE Key structure is used for transporting keys outside of COSE messages. This document extends the way that keys can be identified and transported by providing attributes that refer to or contain X.509 certificates.

Contributing to this document

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. Instructions are on that page as well. Editorial changes can be managed in GitHub, but any substantial issues need to be discussed on the COSE mailing list.

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 February 19, 2020.
1. Introduction

In the process of writing [RFC8152] discussions where held on the question of X.509 certificates [RFC5280] and if there was a needed to provide for them. At the time there were no use cases presented that appeared to have a sufficient need for these attributes. Since that time a number of cases where X.509 certificate support is necessary have been defined. This document provides a set of attributes that will allow applications to transport and refer to X.509 certificates in a consistent manner.

Some of the constrained device situations are being used where an X.509 PKI is already installed. One of these situations is the 6tish environment for enrollment of devices where the certificates are installed at the factory. The [I-D.selander-ace-cose-ecdhe] draft was also written with the idea that long term certificates could be used to provide for authentication of devices and uses them to establish session keys. A final scenario is the use of COSE as a
messaging application where long term existence of keys can be used along with a central authentication authority. The use of certificates in this scenario allows for key management to be used which is well understood.

Example COSE messages for the various headers defined below can be found at https://github.com/cose-wg/Examples. THIS IS NOT YET DONE BUT SHOULD BE COMING NOT LONG AFTER THE F2F MEETING.

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.

1.2. Open Questions

Should we define an extended key usage?
Revocation info?

2. X.509 COSE Headers

The use of X.509 certificates allows for an existing trust infrastructure to be used with COSE. This includes the full suite of enrollment protocols, trust anchors, trust chaining and revocation checking that have been defined over time by the IETF and other organizations. The key structures that have been defined in COSE currently do not support all of these properties although some may be found in COSE Web Tokens (CWT) [RFC8392].

It is not necessarily expected that constrained devices will fully support the evaluation and processing of X.509 certificates, it is perfectly reasonable for a certificate to be assigned to a device which it can then provide to a relying party along with a signature or encrypted message, the relying party not being a constrained device.

Certificates obtained from any of these methods MUST still be validated. This validation can be done via the PKIX rules in [RFC5280] or by using a different trust structure, such as a trusted certificate distributor for self-signed certificates. The PKIX validation includes matching against the trust anchors configured for the application. These rules apply to certificates of a chain length of one as well as longer chains. If the application cannot establish a trust in the certificate, then it cannot be used.
The header attributes defined in this document are:

**x5bag:** This header attribute contains a bag of X.509 certificates. The set of certificates in this header are unordered and may contain self-signed certificates. The certificate bag can contain certificates which are completely extraneous to the message. (An example of this would be to carry a certificate with a key agreement key usage in a signed message.) As the certificates are unordered, the party evaluating the signature will need to do the necessary path building. Certificates needed for any particular chain to be built may be absent from the bag.

As this header element does not provide any trust, the header attribute can be in either a protected or unprotected header attribute.

This header attribute allows for a single or a bag of X.509 certificates to be carried in the message.

* If a single certificate is conveyed, it is placed in a CBOR bstr.

* If multiple certificates are conveyed, a CBOR array of bstrs is used. Each certificate being in its own bstr.

**x5chain:** This header attribute contains an ordered array of X.509 certificates. The certificates are to be ordered starting with the certificate containing the end-entity key followed by the certificate which signed it and so on. There is no requirement for the entire chain to be present in the element if there is reason to believe that the relying party will already have it. This means that the relying party is still required to do path building, but that a candidate path is proposed in this attribute.

As this header element does not provide any trust, the header attribute can be in either a protected or unprotected header attribute.

This header attribute allows for a single or a chain of X.509 certificates to be carried in the message.

* If a single certificate is conveyed, it is placed in a CBOR bstr.
* If multiple certificates are conveyed, a CBOR array of bstrs is used. Each certificate being in it’s own slot.

**x5t:** This header attribute provides the ability to identify an X.509 certificate by a hash value. The attribute is an array of two elements. The first element is an algorithm identifier which is an integer or a string containing the hash algorithm identifier. The second element is a binary string containing the hash value.

As this header element does not provide any trust, the header attribute can be in either a protected or unprotected header attribute.

For interoperability, applications which use this header attribute MUST support the hash algorithm ‘sha256’, but can use other hash algorithms.

**x5u:** This header attribute provides the ability to identify an X.509 certificate by a URI. The referenced resource can be any of the following media types:

* application/pkix-cert [RFC2585]
* application/pkcs7-mime; smime-type="certs-only" [RFC8551]

As this header attribute implies a trust relationship, the attribute MUST be in the protected attributes.

The URI provided MUST provide integrity protection and server authentication. For example, an HTTP or CoAP GET request to retrieve a certificate MUST use TLS [RFC8446] or DTLS [I-D.ietf-tls-dtls13]. If the certificate does not chain to an existing trust anchor, the certificate MUST NOT be trusted unless the server is configured as trusted to provide new trust anchors. This will normally be the situation when self-signed certificates are used.

The header attributes are used in the following locations:

* COSE_Signature and COSE_Sign0 objects, in these objects they identify the key that was used for generating signature.
* COSE_recipient objects, in this location they may be used to identify the certificate for the recipient of the message.
<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>value type</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x5bag</td>
<td>TBD4</td>
<td>COSE_X509</td>
<td>An unordered bag of X.509 certificates</td>
</tr>
<tr>
<td>x5chain</td>
<td>TBD3</td>
<td>COSE_X509</td>
<td>An ordered chain of X.509 certificates</td>
</tr>
<tr>
<td>x5t</td>
<td>TBD1</td>
<td>COSE_CertHash</td>
<td>Hash of an X.509 certificate</td>
</tr>
<tr>
<td>x5u</td>
<td>TBD2</td>
<td>uri</td>
<td>URI pointing to an X.509 certificate</td>
</tr>
</tbody>
</table>

Table 1: X.509 COSE Headers

Below is an equivalent CDDL [RFC8610] description of the text above.

```
COSE_X509 = bstr / [ 2*certs: bstr ]
COSE_CertHash = [ hashAlg: (int / tstr), hashValue: bstr ]
```

3. X.509 certificates and static-static ECDH

The header attributes defined in the previous section are used to identify the recipient certificates for the ECDH key agreement algorithms. In this section we define the algorithm specific parameters that are used for identifying or transporting the sender's key for static-static key agreement algorithms.

These attributes are defined analogously to those in the previous section. There is no definition for the certificate bag as the same attribute would be used for both the sender and recipient certificates.

- **x5chain-sender**: This header attribute contains the chain of certificates starting with the sender’s key exchange certificate. The structure is the same as ‘x5bag’.
- **x5t-sender**: This header attribute contains the hash value for the sender's key exchange certificate. The structure is the same as ‘x5t’.
- **x5u-sender**: This header attribute contains a URI for the sender’s key exchange certificate. The structure and processing are the same as ‘x5u’.
<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Type</th>
<th>Algorithm</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x5t-sender</td>
<td>TBD</td>
<td>COSE_CertHash</td>
<td>ECDH-SS+HKDF-256, ECDH-SS+HKDF-512,</td>
<td>Thumbprint for the</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ECDH-SS+A128KW, ECDH-SS+AES192KW, ECDH-SS+AES256KW</td>
<td>senders X.509</td>
</tr>
<tr>
<td>x5u-sender</td>
<td>TBD</td>
<td>uri</td>
<td>ECDH-SS+HKDF-256, ECDH-SS+HKDF-512, ECDH-SS+A128KW, ECDH-SS+AES192KW, ECDH-SS+AES256KW</td>
<td>URI for the senders X.509 certificate</td>
</tr>
<tr>
<td>x5chain-sender</td>
<td>TBD</td>
<td>COSE_X509</td>
<td>ECDH-SS+HKDF-256, ECDH-SS+HKDF-512,</td>
<td>static key X.509 certificate chain</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ECDH-SS+A128KW, ECDH-SS+AES192KW, ECDH-SS+AES256KW</td>
<td></td>
</tr>
</tbody>
</table>

Table 2: Static ECDH Algorithm Values

4. IANA Considerations

4.1. COSE Header Parameter Registry

IANA is requested to register the new COSE Header items in Table 1 in the "COSE Header Parameters" registry.

4.2. COSE Header Algorithm Parameter Registry

IANA is requested to register the new COSE Header items in Table 2 in the "COSE Header Algorithm Parameters" registry.

5. Security Considerations

Establishing trust in a certificate is a vital part of processing. Trust cannot be assumed whenever a new self-signed certificate appears on the client, instead a well defined process is required. One common way for a new trust anchor to be added (or removed) from a device is by doing a new firmware upgrade.

In constrained systems, there is a trade-off between the order of checking the signature and checking the certificate for validity. Validating certificates can require that network resources be accessed in order to get revocation information or retrieve...
certificates during path building. Doing the network access can consume resources dealing with power and network bandwidth. On the other hand, an oracle can potentially be built based on if the network resources are only accessed if the signature validation passes. In any event, both the signature and certificate validation MUST be checked before acting on any requests.

As called out in the COSE algorithms document [I-D.ietf-cose-rfc8152bis-algs] basic checking on the keys in a certificate needs to be performed prior to using them. These can include validating that points are on curves for elliptical curve algorithms and that sizes of keys are acceptable for RSA. The use of unvalidated keys can lead either to loss of security or excessive consumption of resources.

6. References

6.1. Normative References


6.2. Informative References


Author’s Address

Jim Schaad
August Cellars

Email: ietf@augustcellars.com